Color sequencing

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
(Dr. rer. nat.)

genehmigte Dissertation

vorgelegt von

Dipl.–Math.
Thomas Epping

geboren am 15.03.1974 in Kirchhellen (jetzt Bottrop)

Gutachter: Prof. Dr. Winfried Hochstättler
Gutachter: Prof. Dr. Sven O. Krumke
Gutachter: Prof. Dr. Winfried Kurth

Tag der mündlichen Prüfung: 28.10.2004

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig und ohne unzulässige Hilfe angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; dass diese Dissertation noch keiner anderen Fakultät zur Prüfung vorgelegen hat sowie dass sie, abgesehen von den unten angegebenen Teilpublikationen, noch nicht veröffentlicht worden ist und ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen der geltenden Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Prof. Dr. Winfried Hochstättler betreut worden.

Teile dieser Dissertation sind bereits in [EHO01], [EH03a], [EH03b], [EHL03] und [EHO04] veröffentlicht.

Teile dieser Dissertation sind in [BEH03] zur Veröffentlichung eingereicht.

The tires are the things on your car
That make contact with the road
The car is the thing on the road
That takes you back to your abode

The tires are the things on your car
That make contact with the road
Bummed is what you are
When you go out to your car and it's been towed

I woke up one morning in November
And I realized I love you
It's not your headlights in front
Your tailpipe or the skylight above you
It's the way you cling to the road
When the wind tries to shove you
I'd never go riding away
And come back home without you

— Phish, "Contact" (1988)

Mein besonderer Dank gilt Winfried Hochstättler für sein Vertrauen in und sein Engagement für mich, nicht nur während der Betreuung dieser Arbeit. Ich habe viel von ihm gelernt.

Zu Dank verpflichtet bin ich außerdem Prof. Dr. Ulrich Faigle und Prof. Dr. Rainer Schrader, unter deren Obhut diese Arbeit begann, sowie Prof. Dr. Sven O. Krumke und Prof. Dr. Winfried Kurth, mit deren Gutachten sie endete. Weiter bin ich der gesamten Arbeitsgruppe Faigle/Schrader des Zentrums für Angewandte Informatik Köln, darunter besonders Peter Oertel, sowie den Kollegen der Brandenburgischen Technischen Universität Cottbus, darunter besonders Georg Sustal und Robert Nickel, für die angenehme gemeinsame Arbeit dankbar.

Schließlich möchte ich mich bei all jenen bedanken, die mich zusätzlich auf verschiedenste Art und Weise unterstützt haben; bei meiner Familie, meinen Freunden und besonders bei Saskia Kersten und Maik A. Linder, die es verstanden haben, mir immer dann neuen Mut zuzusprechen, wenn es nötig war.

Zusammenfassung

Die europäische Automobilindustrie sieht sich, um wettbewerbsfähig zu bleiben, einem zunehmenden Zwang zur Minderung von Produktionskosten gegenüber. Einen Beitrag zu signifikanten Einsparungen leistet die Anwendung des *build-to-order*-Verfahrens, das keine teuren Depots erfordert. Diesem Verfahren steht jedoch der zunehmende Kundenwunsch nach einer reichhaltigen Produktpalette gegenüber. Trotz nahezu täglich wechselnder Auftragszusammenstellungen soll kosteneffizient produziert werden.

Diese Arbeit konzentriert sich auf eine Phase des Produktionsprozesses und betrachtet drei Varianten eines Färbungsproblems aus der Lackierstraße einer Automobilfabrik, in der jede Karosserie in ihrer vorgesehenen Farbe lackiert wird. Da jeder Farbwechsel eine Reinigung der Sprühroboter erfordert (was wiederum die Produktionskosten erhöht), betrachten wir das Problem der Minimierung der Anzahl der Farbwechsel in der Lackierkabine der Lackierstraße. Jede Variante dieses Farbwechselminimierungsproblems tritt in der Praxis auf. Die jeweiligen Lösungsansätze unterteilen diese Arbeit auf natürliche Art und Weise in drei Abschnitte, die theoretische Ergebnisse, Lösungsverfahren und die Beschreibung einer Anwendung enthalten.

Der Hintergrund des ersten Abschnitts ist die Möglichkeit des kurzfristigen Farbtauschs zwischen ansonsten identischen Modellen, den die Automobilindustrie derzeit untersucht. Die Repräsentation von jedem Modell als Buchstabe eines Alphabets liefert das Problem der Minimierung von Farbwechseln in einem gegebenen Wort, wobei ein Farbtausch nur zwischen gleichen Buchstaben erlaubt ist. In Zusammenarbeit mit Peter Oertel, Marco E. Lübbecke und Paul S. Bonsma klassifizieren wir dieses neue kombinatorische Problem bezüglich seiner Komplexität. Wir zeigen insbesondere, dass das Problem in seiner allgemeinen Formulierung selbst dann $\mathcal{NP}$-vollständig ist, wenn das Wort entweder nur zwei Farben oder zwei Buchstaben enthält. Sind sowohl die Anzahl der Farben als auch die Anzahl der Buchstaben beschränkt, so kann das Problem mittels dynamischer Programmierung gelöst werden. Von dem allgemeinen Fall aus wenden wir uns strukturierten Instanzen zu und leiten untere und obere Schranken für die minimale Anzahl von Farbwechseln her. Allerdings zeigen wir, dass sogar eine sehr eingeschränkte Version des Problems noch $\mathcal{APX}$-schwer ist. Wir reformulieren diese eingeschränkte Version als ein Kürzestes-Weg-Problem in einer speziellen Klasse von binären Matroiden. Als Konsequenz daraus zeigen wir, dass Instanzen, die nicht sowohl einen F_7- und F_7^*-Minor enthalten, in polynomieller Zeit lösbar sind. Insbesondere sind reguläre Instanzen durch ein lineares Programm lösbar. Für Instanzen, die das Dual eines MaxFlow-MinCut-Matroids sind, diskutieren wir Konsequenzen der dualisierten MaxFlow-MinCut-Theorie.

Der zweite Abschnitt stellt Lösungsansätze für eine Strategie der Farbwechsel-

minimierung bereit, die einen vor der Lackierstraße installierten Linienspeicher benutzt. Linienspeicher sind ein gebräuchliches Werkzeug in der Automobilindustrie, weil sie einen einfachen und kosteneffizienten Weg bieten, Farbblöcke in der Lackierstraße durch eine geeignete Einlagerung und Entnahme von Farben zu erzeugen. Wir betrachten zwei *offline*-Strategien und eine *online*-Strategie, die auf einen Linienspeicher zur Farbwechselminimierung in der Lackierstraße angewendet werden können. Die erste *offline*-Strategie berücksichtigt nur die Entnahme von Farben und adaptiert ein dynamisches Programm für das *multiple sequence alignment*-Problem aus der Molekularbiologie. Das Ergebnis ist ein neues und exaktes Lösungsverfahren, das sich, obwohl das *multiple sequence alignment*-Problem $\mathcal{NP}$-vollständig ist, als hocheffizient für typische Anwendungsinstanzen erweist. Die zweite *offline*-Strategie ist eine Verallgemeinerung der ersten und berücksichtigt sowohl das Einlagern als auch die Entnahme von Farben. Die einzulagernden Farben für den Linienspeicher werden dabei in einem Vorausschaubereich gehalten. Wir präsentieren ein dynamisches Programm, das alle zulässigen Zustände des Linienspeichers mit Hilfe von zwei Elementaroperationen enumeriert. Das resultierende Verfahren hat jedoch eine gewaltige Komplexität und ist auch nach Verwendung von verschiedenen Methoden, die die Laufzeit einer Implementierung drastisch beschleunigen, nur auf kleine Instanzen anwendbar. Es ist jedoch nicht nur von theoretischem Interesse, weil wir Elemente daraus für eine *online*-Simulation zur Farbeinlagerung und -entnahme benutzen. Dieses Mal benutzen wir nur eine Elementaroperation, was näher an der Praxis orientiert ist. Während die erste Hälfte des Vorausschaubereichs verarbeitet wird, wird für die zweite eine optimale Strategie zur Farbeinlagerung und -entnahme berechnet. Diese partiellen Lösungen können zu einer Lösung für einen kompletten Produktionstag zusammengefügt werden und bereits ein *greedy*-Zusammenfügen führt zu einer signifikanten Erhöhung der Farbblockgröße in der Lackierstraße. Insbesondere kann die Simulation leicht an praktische Anforderungen wie Speicherplatzbedarf und die Einhaltung von Taktzeiten angepasst werden. Für alle Verfahren geben wir Rechenergebnisse auf Zufallsinstanzen, die Diskussionen ihrer Vor- und Nachteile begleiten.

Die Beschreibung eines Projekts in Zusammenarbeit mit der Ford-Werke AG bildet den Inhalt des dritten Abschnitts. Die Strategie zur Farbwechselminimierung ähnelt der Strategie des kurzfristigen Farbtauschs aus dem ersten Abschnitt. Wir berücksichtigen jedoch die zusätzliche Tatsache, dass die meisten Automobilfabriken nicht nur eine, sondern gewöhnlicherweise zwei oder drei parallele Lackierkabinen haben. Dies ermöglicht es uns, durch ein geeignetes *routing* Farbblöcke innerhalb jeder Lackierkabine zu bilden, sofern identische Lackfarben nahe beieinander an dem Verzweigungspunkt zu den Lackierkabinen ankommen. Dies ist von besonderer Bedeutung, weil unvermeidbare Produktionsfehler das Beibehalten von Farbblöcken während des Produktionsprozesses verhindern. Unser Lösungsansatz erweitert bekannte Lösungsansätze in mehrfacher Hinsicht.

Insbesondere ist sein Rahmen nicht auf die Lackierstraße beschränkt, sondern umfasst den kompletten Produktionsprozess, was zu mehreren zusätzlichen Optimierungszielen und Nebenbedingungen für andere Produktionsphasen führt, die respektiert werden müssen. Die Möglichkeit eines geeigneten *routing* von Aufträgen wird in einem *clustering*-Schritt berücksichtigt, der zu einer Häufung von Aufträgen mit gleichen Lackfarben in bestimmten Bereichen der Produktionssequenz führt. Das unvermeidbare Auftreten von Produktionsfehlern wird durch die Einführung von deterministischen Auftragsverspätungen antizipiert. Die Kombination von diesen und weiteren Konzepten führt zu einem effizienten Lösungsverfahren, das derzeit in allen europäischen Werken der Ford-Werke AG eingesetzt wird.

Contents

Part I

Introduction

Chapter 1

Motivation

In the European automobile industry, competition has increased over the past years. Fast and easy communication via e-mail and the internet, the single currency, and an increasing price consciousness of consumers raise the pricing pressure and call for savings in all areas of supply, production, and delivery. The extend of investment in research and development (R&D) may act as an indicator for the interest in new solutions and innovations in this major industry. The German automobile industry, for example, has invested more than 15 billion € in R&D in the year 2002, about one third of the entire investments of the German industry in this area (see [VDA03]).

One reason that complicates the reduction of production costs is the need for a rich product variety, as only a rich assortment of models with various characteristics allows flexibility with regard to individual customer demands and thus increases competitiveness. As a consequence, cars are built to order instead of being built to stock to avoid the need for large depots.

Figure 1.1: Automobile production planning.

The build-to-order policy presumes the collection of customer orders over a specific period (see Figure 1.1). Afterwards, these orders are assigned to production days in an order segmentation phase. Order segmentation results in a variety of models that have to be produced. This variety changes almost daily. The final stage of production planning is the order sequencing that fixes the production sequence for a day.

1.1 Automobile order sequencing

Each order is specified by a set of commodities that define the characteristics of a customers' request. A commodity may be, for example, the number of doors or the engine type as well as the latest production day (we ignore, however, such non-hardware commodities as they are not of interest for the order sequencing). Commodities are evaluated and applied as an order passes through various production shops (see Figure 1.2). As each production shop requires a specific commodity sequence to run efficiently, the order sequencing has a great impact both on production quality and production costs.

Figure 1.2: Automobile production shops.

It is common practice to compute a global sequencing of all orders of a given production day that conforms mainly to the needs of the assembly shop where the most significant savings can be obtained (see [Spi02]). However, it is very unlikely that larger parts of a sequence remain unchanged during the production process. There exist lots of eventualities for manufacturing errors at each production shop and each error in an order leads to a production shop rerun and thereby to a perturbation of the intended production sequence. On this account a short-term local sequencing in front of each production shop is reasonable and increases production flexibility. The goal of local sequencing may be either the restoration of the global sequence or a sequencing that suits the respective production shop even better than the global sequence.

1.2 Order sequencing for a paint shop

In contrast to the huge amount of literature available on order sequencing for the assembly shop, only very little effort has been spent on order sequencing for the paint shop, although significant profits can be achieved here, too.

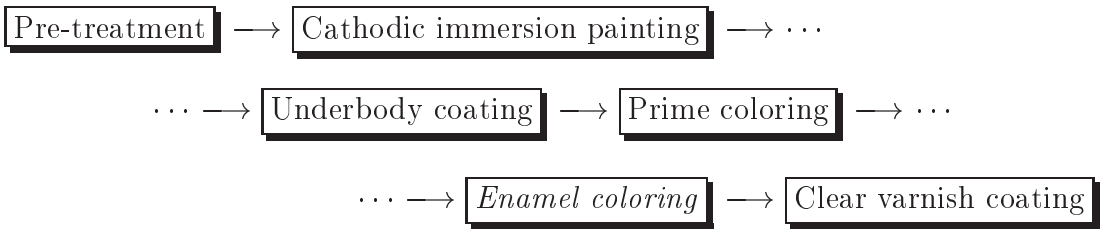


Figure 1.3: Phases of car body painting (see [Spi02]).

A paint shop is typically divided into several booths (see Figure 1.3). We are interested in the enamel booth in which a car body gets its designated enamel color. Within this booth, a color change occurs whenever two consecutive car bodies have to be colored differently.

For each color change the color jets of the spray robots have to be cleaned. While the setup time of spray robots for this operation is negligible, the setup costs of spray robots range from 10 € to 30 € per color change. Typically, colors have to be changed after every second or, at the very latest, after every third order. Assuming a yearly production of approximately 200,000 orders per plant, even a small decrease in the number of color changes leads to a significant reduction of production costs (see [Spi02]).

Thus, we consider an objective function that is easy to state: *Minimize the number of color changes in the enamel booth.* As a positive side effect, the minimization of the number of color changes goes along with the reduction of water pollution caused by the cleaning of the spray robots.

Figure 1.4: Enamel coloring in 1978 [AKG].

1.3 Color change minimization strategies

The color change minimization problem in the enamel booth can be tackled by several strategies. Each strategy results in a different variant of the problem with its specific characteristics. Color changes can be minimized either within the complete production sequence before the start of production or within a short subsequence in front of the paint shop only. We have already mentioned in Section 1.1 that a global color change minimization is very unlikely to be met in practice. For a local color change minimization, the problem usually has to be solved online and in real-time (see [Spi02]). Furthermore, it may be possible to use additional hardware in front of the paint shop for a color change minimization. Note that similar strategies can be applied for other production shops, too.

1.3.1 Color interchanges between models

As a production sequence usually has to conform to the needs of the assembly shop (see Section 1.1), it is necessary to minimize the number of color changes within the production sequence without worsening its quality with respect to the

assembly shop. In other words, if we identify orders that match with respect to all commodities except the enamel color, we are allowed to interchange the enamel color between identical orders only. This can be done in a greedy or, relating to the current subsequence in front of the paint shop, in an optimal fashion.

In the automobile industry, there is currently a trend that enables the improvement of a sequence with respect to a selected commodity by allowing commodity interchanges. This concept increases production flexibility and can be realized by the use of online programmable micro chips attached to each order.

1.3.2 Color resorting by an interim storage system

Another strategy of color change minimization uses an interim storage system for a permutation of the orders of the current subsequence in front of the paint shop. Color resorting is done by a suitable storage or retrieval of orders on or from the interim storage system. The efficiency of an interim storage system is determined by its dimension as well as by storage and retrieval strategies, which can either be heuristics or exact approaches.

Currently, it is common practice to use interim storage systems. Note that, at the same time, interim storage systems provide a safety stock for production.

1.3.3 Color clustering

The color change minimization strategies in Section 1.3.1 and Section 1.3.2 assume only one enamel booth exists. In practice, two or three enamel booths may be arranged parallel to each other, which yields the additional opportunity to build color blocks within each enamel booth by a suitable routing of orders.

In view of unavoidable manufacturing errors and the resulting production sequence perturbation (see Section 1.1), this leads to a strategy of color change minimization that contains aspects of both strategies described in Section 1.3.1 and Section 1.3.2. This time an interim storage system is used to recover the original production sequence at the paint shop after its previous perturbation by manufacturing errors. Before the distribution of orders among the enamel booths (which usually follows a fixed pattern to guarantee an even occupancy and to avoid the need of buffers), enamel colors of identical orders are interchanged to build color blocks within each enamel booth. The efficiency of this approach can be increased if the original sequence contains clusters of orders with the same enamel color.

1.4 Outline

Each part of this thesis is devoted to a subsection of Section 1.3 and deals with the combinatorial problems brought up by the corresponding realities.

Part II deals with color change minimization by enamel color interchanges as described in Section 1.3.1. We give a mathematical formalization of this problem and prove its $\mathcal{NP}$-completeness before we describe a dynamic program for its solution. For structured instances, we state lower and upper bounds for the minimal number of color changes. Then we focus on a very restricted version of the problem which, as we show, is still $\mathcal{APX}$-hard. We reformulate the restriction as a shortest circuit problem in a certain class of binary matroids and investigate the consequences of this reformulation. These include the solvability of specific instances in this class in polynomial time and an application of MaxFlow-MinCut duality.

Part III considers the use of a widely spread interim storage system, more specific a line storage system, for color change minimization as described in Section 1.3.2. We develop a new and exact solution algorithm for the exclusive retrieval of orders from a line storage system as well as for the simultaneous storage and retrieval of orders. We use the latter to propose a new simulation model for color change minimization in a paint shop. All solution algorithms are illustrated by computational results.

Part IV reports on a real-world project in cooperation with the Ford Motor Company that implements the color change minimization strategy described in Section 1.3.3. We point out the constraints that arise from production shops other than the paint shop and that have to be taken into consideration for the computation of a high-quality production sequence. We propose a multi-stage solution algorithm which is currently used by the Ford Motor Company for automobile production sequencing in all plants across Europe.

Chapter 2

Preliminaries

This chapter contains a collection of basic definitions and theorems that are used in this thesis. Our notation is fairly standard. For further details see [Lot97] (for Section 2.1), [PY91] (for Section 2.2), [Gol80] (for Section 2.3 and Section 2.4), and [Oxl92, Tru92] (for Section 2.5).

2.1 Words and word colorings

An *alphabet* Σ is a finite set of elements, which are also called *letters*. A *word* or *sequence* $w = (w_1, \ldots, w_n)$ over an alphabet Σ is a finite ordered multiset with $w_i \in \Sigma$ for $i = 1, \ldots, n$ (we may write $w \in \Sigma^n$ for short). For example, stacks and queues can be interpreted as words. The *length* of w equals the number of letters of w and is denoted by $|w| := n$. The number of occurrences of a letter $\sigma \in \Sigma$ in w is denoted by $|w|_\sigma := |\{w_i \in w : w_i = \sigma \text{ for } i = 1, \ldots, n\}|$. Two letters w_i and w_j with $i < j$ of a word w are called *consecutive* if and only if $1 < i+1 = j \leq n$. A *subsequence* w' of w is any sequence of consecutive letters of w. The *reverse* of a word $w = (w_1, \ldots, w_n)$ is defined by $w^{-1} := (w_n, \ldots, w_1)$, and a word w with $w = w^{-1}$ is called a *palindrom*. Two words $w = (w_1, \ldots, w_n)$ and $w' = (w'_1, \ldots, w'_n)$ are called *equivalent* if there exists a permutation $\tau : \Sigma \to \Sigma$ such that $\tau(w') = w$ or $\tau(w') = w^{-1}$, where $\tau(w') := (\tau(w'_1), \ldots, \tau(w'_n))$.

Let w be a word over an alphabet Σ and c be a word over an alphabet C, such that $\Sigma \cap C = \emptyset$ and $|w| = |c| = n$. We call c a *coloring* of w, and c_i is called the *color* of w_i. A sequence of consecutive, identically colored letters is called a *color block*. Given a coloring c of a word w, we define

$$b_i := \begin{cases} 0 & \text{, if } c_i = c_{i+1} \\ 1 & \text{, otherwise} \end{cases}$$

for $i = 1, \ldots, n-1$. If consecutive letters w_i and w_{i+1} of w are colored differently (i. e. if $b_i = 1$), we say that there is a *color change* between positions i and $i+1$. If it is unambiguous, we also say that there is a color change between letters w_i and w_{i+1}.

The value $\gamma(w;c) := \sum_{i=1}^{n-1} b_i$ is called the *number of color changes* within c or the number of color changes within w. We may also write $\gamma(w)$ if it is clear to which coloring of w we refer to. We extend this notation to $\gamma^*(w;c) := \min\{\gamma(w;\tau(c)) : \tau(w) = w\}$, where the minimum is taken over all permutations $\tau : \{1,\ldots,n\} \to \{1,\ldots,n\}$ and $\tau(c) := (c_{\tau(1)}, \ldots, c_{\tau(n)})$. Again, we may write $\gamma^*(w)$ instead of $\gamma^*(w;c)$ for short, and c is called an *optimal coloring* of w in this context.

We usually use the alphabet $\Sigma = \{A, B, C, \ldots\}$ for words and illustrate the coloring of a word by colored letters.

2.2 Approximation algorithms

A *ρ-approximation algorithm* for a minimization (maximization) problem is an algorithm that computes a solution with value at most (least) ρk for every instance with optimal solution value k, where $\rho = 1+\epsilon$ ($\rho = 1-\epsilon$) with $\epsilon > 0$. A *polynomial time approximation scheme (PTAS)* for a minimization (maximization) problem Π is a family of polynomial time $(1+\epsilon)$-approximation ($(1-\epsilon)$-approximation) algorithms for Π, one for every $\epsilon > 0$. The problem class $\mathcal{APX}$ is the class of minimization (maximization) problems for which a polynomial time ρ-approximation algorithm exists for some $\rho > 1$ ($0 < \rho < 1$). An optimization problem is called $\mathcal{APX}$-hard if the existence of a PTAS for this problem would imply that for every problem in $\mathcal{APX}$ a PTAS exists.

Theorem 2.1 (Arora et al. [A+98]) *If there exists a PTAS for an $\mathcal{APX}$-hard problem, then $\mathcal{P}$=$\mathcal{NP}$.* ■

It is agreed that $\mathcal{P}$=$\mathcal{NP}$ is quite unlikely.

Let Π_1 and Π_2 be optimization problems. We say that Π_1 *L-reduces* to Π_2 if there exist polynomial time algorithms A_1 and A_2, and constants $\alpha \in \mathbb{R}_{>0}$ and $\beta \in \mathbb{R}_{>0}$ such that the following conditions hold for every instance x of Π_1.

1. $\mathrm{opt}_{\Pi_2}(A_1(x)) \leq \alpha \cdot \mathrm{opt}_{\Pi_1}(x)$.
2. Given any feasible solution of $A_1(x)$ with solution value c_2, algorithm A_2 produces a feasible solution of x with solution value c_1, and $|\mathrm{opt}_{\Pi_1}(x) - c_1| \leq \beta \cdot |\mathrm{opt}_{\Pi_2}(A_1(x)) - c_2|$.

Here $\text{opt}_\Pi(x)$ denotes the optimal solution value of a feasible solution of x, where x is an instance of an optimization problem Π.

2.3 Graphs

An *undirected graph* is an ordered pair $G = (V, E)$ where V is a finite set of *vertices* and $E \subseteq (V \times V) \setminus \{(v, v) : v \in V\}$ is a set of undirected *edges*. We may write ij for edges instead of (i, j) for short. If $G = (V, E)$ is an undirected graph and $ij \in E$ is an edge, the vertices $i \in V$ and $j \in V$ are called *adjacent*, and both i and j are called *incident* with ij.

We denote the *degree* of a vertex $v \in V$ by $d(v) := |\{ij \in E : v \in \{i, j\}\}|$ and the *maximal degree* of G is defined by $\max\{d(v) : v \in V\}$. G is called a *cubic graph* if $d(v) = 3$ holds for all $v \in V$. A *vertex cover* of $G = (V, E)$ is a set $C \subseteq V$ such that every edge $e \in E$ is incident with at least one vertex of C. A *stable set* in G is a set $S \subseteq V$ such that no two vertices in S are adjacent.

Let $G = (V, E)$ denote an undirected graph. The graph $G-v := (V\setminus\{v\}, E\setminus\{ij \in E : v \in \{i, j\}\})$ denotes the graph obtained from G by removing vertex v and all edges incident with v. The graph $G + ij := (V, E \cup ij)$ denotes the graph obtained by adding an edge between $i \in V$ and $j \in V$.

Edges with a direction assigned to them are called *arcs*. An arc from $i \in V$ to $j \in V$ is denoted by (i, j). In this case i is an *in-neighbor* of j and j is an *out-neighbor* of i. If each edge of a graph G has a direction assigned to it, G is called a *directed graph* and denoted by $G = (V, A)$. Let $G = (V, A)$ be a directed graph and $H = (V, E)$ be an undirected graph. If $|A| = |E|$ and the implication $(i, j) \in A \Rightarrow ij \in E$ holds, then A is called an *orientation* of H.

We introduce the following notation for directed graphs only. It can, however, be easily modified and applied to undirected graphs as well. Let $G = (V, A)$ be a directed graph. A graph $G' = (V', A')$ is called a *subgraph* of G if $V' \subseteq V$ and $A' \subseteq A$ holds. Given a subset $V' \subseteq V$, the graph $G' = (V', \{(i, j) \in A : i \in V' \text{ and } j \in V'\})$ is called the subgraph of G *induced* by V'. A sequence $(v_1, \ldots, v_k)$ of pairwise distinct vertices with $(v_i, v_{i+1}) \in A$ for $i = 1, \ldots, k-1$ is called a *path*. A *cycle* is the union of a path $(v_1, \ldots, v_k)$ with the arc (v_k, v_1). A *weighted graph* is a directed graph $G = (V, A; w)$ with a weight function w. The graph G is called a *vertex-weighted* directed graph if $w : V \to \mathbb{Q}_{\geq 0}$. The graph G is called an *arc-weighted* directed graph if $w : A \to \mathbb{Q}_{\geq 0}$. It is well known that $\mathbb{Q}$ may be replaced by $\mathbb{N}$ after the multiplication of all weights with their common denominator.

2.4 Sets

Let S be a finite set (we only consider finite sets). We denote the *power set* of S by $2^S := \{X : X \subseteq S\}$. A family $\mathcal{S} \subseteq 2^S$ of subsets of a set S is called a *clutter* if the implication $X \subseteq Y \Rightarrow X = Y$ holds for all $X, Y \in \mathcal{S}$, i. e. a clutter is a family of sets where no set properly contains another set.

If I is a set of intervals in $\mathbb{R}$, the *interval graph* $G_I = (I, E)$ is the undirected graph defined by $I_\sigma I_\tau \in E :\Leftrightarrow I_\sigma \cap I_\tau \neq \emptyset$.

Let $\mathcal{S} \subseteq 2^S$ be a family of subsets of a set S and $w : \mathcal{S} \to \mathbb{Q}_{\geq 0}$ be a weight function on $\mathcal{S}$. A set $Q \in \mathcal{S}$ is called *maximal*, if there does not exist a set $R \in \mathcal{S}$ with $Q \subsetneq R$. A set $Q \in \mathcal{S}$ is called *maximum*, if $w(Q) \geq w(R)$ for all $R \in \mathcal{S}$. The terms *minimal* and *minimum* are defined analogously.

A *partition* P of a set S is a decomposition of S into $m < \infty$ pairwise disjoint sets, i. e. $P = P_1 \cup \ldots \cup P_m$ with $P_i \cap P_j = \emptyset$ for $1 \leq i < j \leq m$ and $\bigcup_{i=1}^{m} P_i = S$.

2.5 Matroids

A *matroid* M on a set E is an ordered pair $M = (E, \mathcal{I})$, where E is a finite set and $\mathcal{I}$ is a multiset of subsets of E that satisfies the following axioms.

1. $\emptyset \in \mathcal{I}$.
2. If $I \in \mathcal{I}$ and $I' \subseteq I$, then $I' \in \mathcal{I}$.
3. If $I_1, I_2 \in \mathcal{I}$ with $|I_1| < |I_2|$, then there exists $e \in I_2 \backslash I_1$ such that $(I_1 \cup e) \in \mathcal{I}$.

The elements of $\mathcal{I}$ are called the *independent sets* of M. Subsets of E that are not in $\mathcal{I}$ are called *dependent*. A *circuit* of M is a minimal dependent set of M. A maximal independent set of a matroid M is called a *base* of M. All bases of M have the same cardinality.

Let A be an $(m \times n)$-matrix over a field $\mathbb{F}$. The *vector matroid* $M[A] = M(E, \mathcal{I})$ is defined by $E := \{1, \ldots, n\}$, and a set $X \subseteq E$ is contained in $\mathcal{I}$ if and only if the columns of A specified by X are linearly independent in the vector space $\mathbb{F}^m$ in terms of linear algebra. If it is clear which matrix we refer to, we may write M instead of $M[A]$ for short.

A matroid M is called *representable* over a field $\mathbb{F}$ if M is isomorphic to $M[A]$ for some matrix A over $\mathbb{F}$, and A is called a *representation* for M over $\mathbb{F}$. We focus on the cases $\mathbb{F} = \mathrm{GF}(2) = \{0, 1\}$ and $\mathbb{F} = \mathbb{R}$. A *binary matroid* is a matroid which is representable over $\mathrm{GF}(2)$. We may assume that, after a sequence of

elementary row and column operations, a representation of a vector matroid M over $\mathbb{F}$ is given by its *standard representative matrix* of the form (I, B), where I denotes the identity matrix and B denotes a matrix of appropriate dimension over $\mathbb{F}$. We may write $M[I, B]$ instead of $M[(I, B)]$ for short.

Let $M = M[A]$ denote a binary vector matroid represented by a matrix A over GF(2). We call M *regular* if the entries of A can be signed so that A is represented by a *totally unimodular* matrix over $\mathbb{R}$, where a matrix is called totally unimodular if every of its square submatrices has a determinant in $\{0, \pm 1\}$. A binary matroid $M[I, B]$ is regular if and only if there exists a totally unimodular signing of B.

Theorem 2.2 (Camion [Cam65]) *A totally unimodular signing of B can be found in polynomial time and is unique up to multiplications of rows or columns of B with -1.* ■

A binary matrix has the *consecutive ones property* for rows (columns) if it columns (rows) can be permuted so that the ones in each row (column) occur consecutively.

Theorem 2.3 (see e. g. [GGL95]) *A matrix with the consecutive ones property for rows is totally unimodular.* ■

Let $M = (E, \mathcal{I})$ be a matroid, $X \subseteq E$, and $\mathcal{I}|X$ defined by $\mathcal{I}|X := \{I \subseteq X : I \in \mathcal{I}\}$. Then $M \setminus (E - X) := (X, \mathcal{I}|X)$ is a matroid which is called the *deletion* of $E - X$ from M. A matroid $N = (E', \mathcal{I}')$ is called a *one-point extension* of M, if M can be obtained from N by the deletion of a single element of E'. In this context, the single element may be the vector (of appropriate dimension) whose entries are all zero resp. one, which we denote by $\vec{0} = (0, \ldots, 0)$ resp. $\vec{1} = (1, \ldots, 1)$.

$$\left(\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right) \qquad \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right)$$

(a) (b)

Figure 2.1: Standard representation matrices of (a) F_7 and (b) F_7^* over GF(2).

Let $M = (E, \mathcal{I})$ be a matroid. The *dual matroid* (or *dual* for short) $M^* = (E^*, \mathcal{I}^*)$ of M is defined by $E^* := E$ and $\mathcal{I}^* := \{E \setminus B : B \text{ is a base of } M\}$. It is common practice to refer to the dual of a matroid by the use of the prefix "co" like, for example, in the words "cocircuit" or "cobase". Let $T \subseteq E$ be a subset of E. The

contraction M/T of T from M is defined by $M/T := (M^* \setminus T)^*$. If (I, B) is a representation for a matroid M, the dual matroid M^* is represented by $(-B^T, I)$.

Figure 2.1 shows the standard representative matrices for the binary matroids known as the *Fano* matroid F_7 and its dual F_7^*, both of which are not representable over fields with a characteristic different from 2. Contractions and deletions on a matroid $M = (E, \mathcal{I})$ can be performed in any order. A matroid N is called a *minor* of M, if N is isomorphic to $M \setminus X/Y$, where $X \subseteq E$ and $Y \subseteq E$ with $X \cap Y = \emptyset$.

Part II

Theory

This part deals with mainly theoretical results that arise from the order sequencing for a paint shop by interchanges of the enamel color between identical models. Given a production sequence, our goal is to improve the sequence with respect to the color change minimization for the paint shop without worsening the quality of the model sequence. As mentioned in Section 1.1, we may interchange two orders if their commodity sets differ in their enamel color only. Automobile manufacturers currently investigate this possibility of building color blocks within a production sequence (see [Spi02]), as it has the advantage that its realization requires almost no additional hardware within the production plant. Note that this approach, however, neglects the significant perturbation of a sequence by manufacturing errors.

The strategy of enamel color interchanges between identical models may be applied in a local or a global fashion. We consider the latter and represent each model by an element of an alphabet Σ so that identical letters represent orders that differ at most in their enamel color. We assume that a production sequence (i. e. a word over Σ) and its initial coloring are given by an arbitrary initial sequencing and consider the objective of color change minimization within the word by color interchanges between identical letters only.

This problem of color change minimization combines aspects of coloring problems with aspects of sequencing problems. Existing theories do not cover all these aspects of the color change minimization problem. For example, there exists a theory of combinatorics on words (see [Lot97]) which deals, among other topics, with the construction of infinite words that posses certain properties and asymptotic results of unavoidable regularities in words. Optimization and coloring aspects, however, are missing. In turn, a formulation of the color sequencing problem as a graph coloring problem misses the sequencing aspect, while a formulation as a shortest path problem disregards the necessity to meet predetermined demands of colored letters.

Our color sequencing problem is therefore a new combinatorial problem. In Chapter 3, we give its general mathematical formulation and present complexity results and a dynamic program for its solution. Afterwards, we turn our focus to instances that have more structure. Chapter 4 contains results on upper bounds for the minimal number of color changes for such instances. We then consider more specific instances in Chapter 5. The complexity of this restricted version of the original color sequencing problem is classified again by a complexity result that shows its $\mathcal{APX}$-hardness. We describe an efficient method to derive a lower bound on the number of color changes of a given instance. Moreover, we show how to transform the color sequencing problem into the problem of finding a shortest circuit in a binary matroid. This approach enables us to apply known theorems of matroid theory which characterize those instances that are polynomially solvable. Furthermore, we discuss consequences of linear programming

duality and MaxFlow-MinCut duality when applied to those instances.

Chapter 3

A paint shop problem for words

Given a word $w \in \Sigma^n$ and a coloring $c \in C^n$ of w, we look for a permutation that minimizes the number of color changes in c and leaves the sequence of letters in w unchanged. The main result of this chapter is the $\mathcal{NP}$-completeness of this problem even if we restrict either $|\Sigma|$ or $|C|$.

Problem 3.1 Paint shop problem for words (PPW)

Instance *An alphabet Σ, a word $w = (w_1, \ldots, w_n)$ over Σ, a color set C, and a coloring $c = (c_1, \ldots, c_n) \in C^n$ of w.*

Question *Find a permutation $\tau : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $\tau(w) = w$ and the number of color changes within $\tau(c)$ is minimized.*

Note that the coloring of a word determines the color reservoirs of its letters. We may specify these reservoirs instead of an explicit color vector.

Definition 3.1 *Given a word $w \in \Sigma^n$ and a coloring $c \in C^n$ of w, we denote the* reservoir *of letter $\sigma \in \Sigma$ in color $c \in C$ by $R(\sigma, c) \in \mathbb{N}_{\geq 0}$, i. e. $R(\sigma, c)$ denotes the number of copies of letter σ in color c, and $\sum_{c \in C} R(\sigma, c) = |w|_\sigma$ holds.*

An alternate perception for the PPW is therefore the synthesis of a predetermined word from restricted reservoirs $R(\sigma, c)$ of colored letters with a minimal number of color changes, where $\sum_{\sigma \in \Sigma} \sum_{c \in C} R(\sigma, c) = |w|$ holds.

Example 3.1 *The word*

$$w = (P, A, T, T, H, E, P, E, T, O, N, T, H, E, H, E, A, D, P, A, T)$$

is a PPW instance with $|C| = 3$ *and* $\Sigma = \{A, D, E, H, N, O, P, T\}$. *A permutation of identical letters of different colors yields*

$$w = (P, A, T, T, H, E, P, E, T, O, N, T, H, E, H, E, A, D, P, A, T)$$

and reduces the number of color changes in w *from* $\gamma(w) = 15$ *to* $\gamma^*(w) = 11$ *(in fact, this is the unique optimal coloring of* w*).*

3.1 Complexity results

Even restricted versions of the PPW are $\mathcal{NP}$-complete. We give reductions that show that the PPW remains $\mathcal{NP}$-complete even if either $|\Sigma| = 2$ or $|C| = 2$ holds. The latter reduction arose from collaboration with Peter Oertel (see [EHO04]).

Theorem 3.1 *The PPW is* $\mathcal{NP}$*-complete for* $|C| = 2$.

Proof. We give a reduction from 3SAT (see [GJ79]).

Problem 3.2 3-Satisfiablity (3SAT)

Instance *A set* V *of variables and a collection* C *of clauses over* V *such that* $|c| = 3$ *for each clause* $c \in C$.

Question *Find a satisfying truth assignment for* C.

Let $C = \{c_1, \ldots, c_m\}$ denote the set of clauses and $V = \{v_1, \ldots, v_n\}$ denote the set of variables of an instance of 3SAT. We construct an instance of the PPW as follows. The word w is a sequence of n *variable blocks*. We explain the construction of a single variable block for a fixed variable v_i in detail, using the colors red and blue. In this proof, we use the terms *characters* and *letters* to distinguish between colored elements (the characters) and uncolored symbols of the alphabet (the letters).

We assume that v_i appears in clauses $c_{i_1}, \ldots, c_{i_k}$ as literals $l_{i_1}, \ldots, l_{i_k}$ and wlog. literals are sorted, such that $l_{i_j} = v_i$ for $j = 1, \ldots, r$ and $l_{i_j} = \bar{v}_i$ for $j = r+1, \ldots, k$. Note that we can always assume $r \geq 1$ and sort the literals independently for each variable block. The variable block of v_i is then a sequence of k *literal blocks* $b_{i_1}, \ldots, b_{i_k}$. The characters we use for building the variable block are given as follows.

- We introduce $4k$ *variable characters* using $2k$ *variable letters* L^i_j $(j = 1, \ldots, 2k)$, each once in red and once in blue. Variable letters differ for each variable block. In the following, we drop their superscript whenever it is clear which variable we refer to.

- We introduce $3m$ *satisfaction testers* using m letters T_s $(s = 1, \ldots, m)$, each once in red and twice in blue. Each letter T_s is associated with a clause c_s. We use the satisfaction testers T_{i_j} $(j = 1, \ldots, k)$, one of each, within the variable block.
- We introduce a *separator letter* Z, available only in blue. We use $k + 1$ *separator characters* within the variable block considered.

We arrange the letters of the variable block as follows. For each literal block we provide four variable letters, as illustrated in Figure 3.1. We precede the first variable letter of a literal block b_{i_j} with ZT_{i_j} for $j = 1, \ldots, r$ and precede the third variable letter of a literal block b_{i_j} with ZT_{i_j} for $j = r + 1, \ldots, k$. Finally, we add a separator Z behind the last variable letter of a variable block.

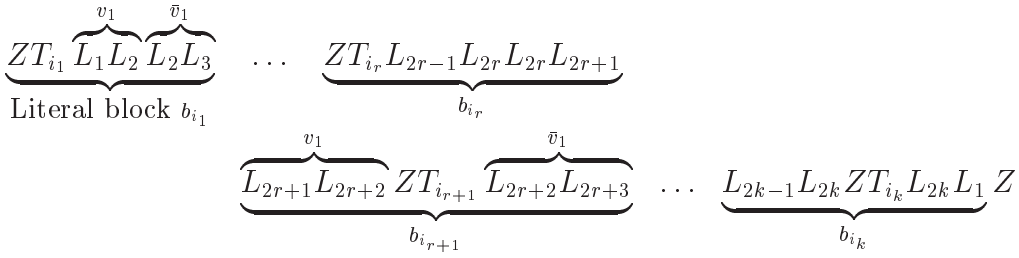


Figure 3.1: The structure of a variable block.

As each variable letter L_j $(j = 1, \ldots, 2k)$ is available only once in each color, it is guaranteed that the two variable characters L_{2j} $(j = 1, \ldots, k)$ in each literal block b_{i_j} have different colors. We claim that two color changes suffice for each literal block.

Claim 3.1 *There exists a satisfying truth assignment for $v_1, \ldots, v_n$ if and only if there exists a coloring of w with exactly $\gamma(w) = 6m$ color changes.*

Proof. Given a satisfying truth assignment, let $c_s = \{l_{s_1}, l_{s_2}, l_{s_3}\}$ be a clause and assume that l_{s_1} satisfies the clause. We then color the associated satisfaction tester T_s in the literal block of l_{s_1} red (and have to color the satisfaction testers in the literal blocks of l_{s_2} and l_{s_3} blue). Additionally, we color the variable letters of the variable block alternatingly (red, red, blue, blue, ...), if the variable related to the variable block is set to `true` in the truth assignment; if the variable is set to `false`, we use the alternating color scheme (blue, blue, red, red, ...).

If our coloring starts with two blue variable characters and we come across a color change between two literal blocks, we always add it to the number of color changes of the preceding one (including the color change between the last variable

character and the final separator). Otherwise, if our coloring starts with two red variable characters and we come across a color change between two literal blocks, we always add it to the number of color changes of the succeeding one. In both cases, this approach results in exactly two color changes within any literal block. Thus, we end up with altogether $6m$ color changes for a coloring of w.

Now, suppose that we are given a coloring with exactly $6m$ color changes. First note that, counting in a similar way as above, we have at least two color changes per literal block. As we know that we have a total of $3m$ literals, we must have exactly two color changes per literal block. This is only possible if the variable letters within each variable block form color blocks of length 2, resulting in a color sequence of either (red, red, blue, blue, ...) or (blue, blue, red, red, ...).

We then assign the value `true` to variables whose variable block starts with two red variable characters and `false` otherwise, and show that we get a satisfying truth assignment. Let $c_s = \{l_{s_1}, l_{s_2}, l_{s_3}\}$ be a clause and assume that the associated satisfaction tester T_s in the literal block of l_{s_1} has been colored red. In order not to give rise to more than two color changes in this literal block, the satisfaction tester must be followed by a red letter. This implies that l_{s_1} is positive if and only if its variable has been set to `true`. Thus, the formula is satisfied. ■

The construction of an instance for the PPW from an instance of 3SAT together with Claim 3.1, which shows how to set the respective questions in correlation, completes our proof. ■

If we bound the size of the alphabet instead of the number of colors, we get a similar result. The proof uses a unary coding and thus only a pseudo-polynomial reduction. Note that this, however, does not cause any problems (see [GJ79], Lemma 4.1, pp. 101).

Theorem 3.2 *The PPW is $\mathcal{NP}$-complete for $|\Sigma| = 2$.*

Proof. We use a pseudo-polynomial reduction from 3-PARTITION (see [GJ79]).

Problem 3.3 3-PARTITION

Instance *A set A of $3m$ elements, a bound $B \in \mathbb{N}_{\geq 0}$, and a size $s(a) \in \mathbb{N}_{\geq 0}$ for each $a \in A$ such that $\frac{B}{4} < s(a) < \frac{B}{2}$ and such that $\sum_{a \in A} s(a) = mB$.*

Question *Find a partition of A into m sets $A_1, \ldots, A_m$ such that, for $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$. (Note that each A_i must therefore contain exactly three elements from A.)*

Let $A = \{a_1, \ldots, a_{3m}\}$ denote the set of elements with size $s(a_1), \ldots, s(a_{3m}) \in \mathbb{N}_{\geq 0}$ and $B \in \mathbb{N}_{\geq 0}$ denote the bound of an instance of 3-PARTITION. We construct an instance of the PPW as follows. For each element a_i of size $s(a_i)$ we

provide $s(a_i)$ times the letter L in color $c(a_i)$, where $c(a_i) \neq c(a_j)$ if $i \neq j$. Additionally, we provide $m-1$ times the letter Z in color c_0. The word w then consists of m *partition blocks* $b_1, \ldots, b_m$ of length B that contain the letter L and are separated by the letter Z (see Figure 3.2).

$$\underbrace{L \ldots L}_{B} Z \underbrace{L \ldots L}_{B} Z \ldots Z \underbrace{L \ldots L}_{B}$$

Figure 3.2: The structure of w.

Clearly, there exists a partition of A into m sets $A_1, \ldots, A_m$ such that $\sum_{a \in A_i} s(a) = B$ for $i = 1, \ldots, m$ if and only if there exists a coloring of w with $\gamma(w) = 4m - 2$ color changes. ■

3.2 Solution by dynamic programming

Any instance of the PPW is solvable in polynomial time if we restrict both $|\Sigma|$ and $|C|$. We now describe a dynamic program which basically passes through the given word w letter by letter from the left to the right. Each optimal coloring of w up to the current position that uses a subset of the letter reservoirs and ends with a specific color is recorded in a different state.

Definition 3.2 *We denote a state of the dynamic program by*

$$S_p = S_p(l_1^1, \ldots, l_{|C|}^1, l_1^2, \ldots, l_{|C|}^2, \ldots, l_1^{|\Sigma|}, \ldots, l_{|C|}^{|\Sigma|}; c) \text{ with } p = \sum_{i,j} l_j^i,$$

where l_j^i counts the occurrence of letter i in color j, and c denotes the color of w_p. A state is feasible *if $l_j^i \leq R(i,j)$ for all i and all j. We denote the* minimal number of color changes *of the partial coloring of w up to position p by $\gamma^*(S_p)$.*

Note that given a state S_{p-1} the letter σ that has to be colored next is determined by $\sigma = w_p$. We apply the dynamic program depicted in Algorithm 3.1.

Theorem 3.3 *Algorithm 3.1 solves an instance of the PPW with a space complexity of $\mathcal{O}(|C|n^{|C||\Sigma|})$ and a time complexity of $\mathcal{O}(|C|^2 n^{|C||\Sigma|})$.*

Proof. We proceed by induction on p and show that $\gamma^*(S_p)$ is the minimal number of color changes that appear if we use the color reservoir recorded in S_p for a coloring of the first p letters of w. The initialization $\gamma^*(S_0) = 0$ for $p = 0$

Algorithm 3.1 A dynamic program for the solution of the PPW.

for all $c \in C$ **do**
  Create $S_0(0,\ldots,0;c)$ and set $\gamma^*(S_0(0,\ldots,0;c)) = 0$
for $p = 1,\ldots,n$ **do**
  for all $c \in C$ **do**
    for all states $S_{p-1}(\ldots,l_c^{w_p},\ldots;d)$ **do**
      if $R(w_p,c) - l_c^{w_p} > 0$ **then**
        Create $S_p = S_p(\ldots,l_c^{w_p}+1,\ldots;c)$
        Set $\gamma^*(S_p) =$
          $\min_{k\neq c}\{\gamma^*(S_{p-1}(\ldots,l_c^{w_p},\ldots;c)), \gamma^*(S_{p-1}(\ldots,l_c^{w_p},\ldots;k))+1\}$
return a state S_n for which $\gamma^*(S_n)$ is minimal

is clearly correct. If $p > 0$, the letter that has to be colored next for creating a state S_p is w_p. We can color w_p in color c only if there is at least one letter w_p in color c left. Thus $R(w_p,c) - l_c^{w_p}$ has to be positive. Then $\gamma^*(S_p)$ is the minimum of $\gamma^*(S_{p-1})$, where in state S_{p-1} the letter w_{p-1} has been colored with the same color as w_p, and $\gamma^*(S_{p-1})+1$, where in state S_{p-1} the letter w_{p-1} has been colored with a different color as w_p, increased by 1 due to the additional color change.

As every l_j^i is bounded from above by $l_j^i \leq n$, we directly get $|C|n^{|C||\Sigma|}$ as an upper bound for the number of states that can be created. The computation of $\gamma^*(S_p)$ requires $\mathcal{O}(|C|)$ steps. ■

Informally speaking, we consider an arc-weighted directed acyclic graph $G = (V,A;w)$ of partial colorings. The vertices of G correspond to the set of all feasible states. An arc $a = (S_{p-1},S_p)$ between states S_{p-1} and S_p occurs if these states have matching values of l_j^i except for one l_{j0}^{i0} and is weighted with 0 or 1, depending on the occurrence of a color change when progressing from S_{p-1} to S_p. Algorithm 3.1 searches for a shortest path from an artificial root to a state S_n within G (see Figure 3.3).

Note that it is sufficient to record l_j^i only for $|C|-1$ colors. We can therefore improve on the complexity of Algorithm 3.1.

Corollary 3.1 *Algorithm 3.1 can be implemented to run with a space complexity of* $\mathcal{O}(|C|n^{(|C|-1)|\Sigma|})$ *and a time complexity of* $\mathcal{O}(|C|^2 n^{(|C|-1)|\Sigma|})$.

Proof. Clearly, $l_{|C|}^i = \sum_{j\in C} R(i,j) - \sum_{j=1}^{|C|-1} l_j^i$ holds for all $i \in \Sigma$. ■

Corollary 3.1 indicates that Algorithm 3.1 is not applicable even for small values of $|\Sigma|$ and $|C|$. Indeed, we show in Chapter 5 that even a very restricted version of the PPW is still $\mathcal{APX}$-hard.

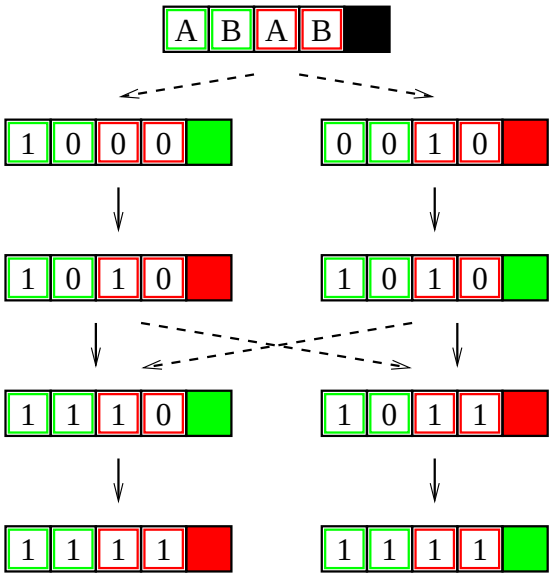


Figure 3.3: Illustration of Algorithm 3.1. A shortest path through G for the PPW instance $w = (A, A, B, B)$ yields an optimal solution $w = (A, A, B, B)$. Dashed arcs are weighted with 0, solid arcs are weighted with 1.

Chapter 4

Regular instances of the PPW

The complexity results on the PPW naturally suggest turning the focus on specific instances of the PPW that appear likely to be accessible for polynomial time solution approaches. Thus, we consider instances whose color reservoirs fulfill a uniformity condition.

Definition 4.1 *Let $k \in \mathbb{N}_{>0}$ be a fixed integer. We call an instance of the PPW* k-regular*, if $R(\sigma, c) = k = \frac{n}{|\Sigma||C|}$ for all letters $\sigma \in \Sigma$ and colors $c \in C$.*

Even the restriction of the PPW to k-regular instances results in an $\mathcal{NP}$-complete problem in general. We delay further complexity results and solution approaches for specific k-regular instances until Chapter 5. In this chapter, we give results and a conjecture on upper bounds for the minimal number of color changes for k-regular instances of the PPW.

Example 4.1 *The word*

$$w = (ABBACBDCADBADCDBABDCACDC)$$

is a 2-regular PPW instance with $|C| = 3$ and $\Sigma = \{A, B, C, D\}$. A permutation of identical letters of different colors yields

$$w = (ABBACBDCADBADCDBABDCACDC)$$

and reduces the number of color changes in w from $\gamma(w) = 17$ to $\gamma^*(w) = 4$.

4.1 Upper bounds

As any k-regular instance has a uniquely determined color reservoir $R(\sigma, c)$, we may denote k-regular instances of the PPW simply by w without causing confusion. We start by giving an upper bound on $\gamma^*(w)$ for the simple case in which both $|\Sigma| = 2$ and $|C| = 2$ hold and we can find a color block of length $\frac{n}{2}$ in w.

Lemma 4.1 *Let w be a k-regular instance of the PPW with $|\Sigma| = |C| = 2$. Then $\gamma^*(w) \leq 2$ holds.*

Proof. We consider a subsequence $w^s := (w_s, w_{s+1}, \ldots, w_{s+\frac{n}{2}-1})$ of w with $|w^s| = 2k = \frac{n}{2}$ and $1 \leq s \leq \frac{n}{2} + 1$ and suppose that $\Sigma := \{x, y\}$. Since $|w^1|_x + |w^{\frac{n}{2}+1}|_x = 2k$, there must exist a t with $1 \leq t \leq \frac{n}{2}+1$ such that $|w^t|_x = k$. As $|\Sigma| = 2$, it follows that $|w^t|_y = k$ as well. We can therefore color all letters of w^t with one color and all letters of $w \setminus w^t$ with the other. Note that $\gamma^*(w) = 1$ if it is possible to choose $t = 1$ or $t = \frac{n}{2} + 1$ and $\gamma^*(w) = 2$ otherwise. ■

Now we consider k-regular instances with $|\Sigma| = 2$ and an arbitrary size of C and use Lemma 4.1 to prove Theorem 4.1 by induction.

Theorem 4.1 *Let w be a k-regular instance of the PPW with $|\Sigma| = 2$. Then $\gamma^*(w) \leq 2(|C| - 1)$ holds.*

Proof. Lemma 4.1 shows that Theorem 4.1 is true for $|C| = 2$. We assume that Theorem 4.1 is true for all values strictly smaller than $|C|$. We again suppose that $\Sigma := \{x, y\}$ and consider a subsequence w^s of w with $|w^s| = 2k = \frac{n}{|C|}$ and $1 \leq s \leq n - \frac{n}{|C|} - 1$. Again, there must exist a t with $|w^t|_x = |w^t|_y = k$, so that we can color all letters of w^t with a single color. The remaining uncolored letters form a k-regular instance $w' = w \setminus w^t$ of length $|w'| = 2k(|C| - 1)$. By our assumption we can color the letters of w' with the remaining $|C| - 1$ colors, so that $\gamma^*(w') \leq 2(|C| - 2)$. We therefore get $\gamma^*(w) \leq 2(|C| - 2) + 2 = 2(|C| - 1)$ for any optimal coloring of w. ■

For a k-regular instance w with an arbitrary size of Σ and $|C| = 2$, we were not able to deduce an upper bound on $\gamma^*(w)$. We conjecture the following.

Conjecture 4.1 *Let w be a k-regular instance of the PPW with $|C| = 2$. Then $\gamma^*(w) \leq |\Sigma|$ holds. Moreover, let w be a k-regular instance of the PPW with an arbitrary size of C. Then $\gamma^*(w) \leq |\Sigma|(|C| - 1)$ holds.*

A linear algebra argument yields the correctness of Conjecture 4.1 for 1-regular instances of the PPW. Furthermore, Example 4.2 shows that the upper bound on $\gamma^*(w)$ given in Conjecture 4.1 is tight if the conjecture is true.

Theorem 4.2 *Let w be a 1-regular instance of the PPW. Then $\gamma^*(w) \leq |\Sigma|(|C|-1)$ holds.*

Proof. We delay the proof until Section 5.6.1. ■

Example 4.2 *Let w be a k-regular instance of the PPW with $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ and w given by*

$$w = (\underbrace{\sigma_1, \ldots, \sigma_1}_{k|C|}, \underbrace{\sigma_2, \ldots, \sigma_2}_{k|C|}, \ldots, \underbrace{\sigma_{|\Sigma|}, \ldots, \sigma_{|\Sigma|}}_{k|C|}).$$

Then $\gamma^*(w) = |\Sigma|(|C|-1)$ holds.

Chapter 5

Shortest paths through two-tone pairs

This chapter deals with 1-regular instances of the PPW with a color set of size $|C| = 2$, so that each letter $\sigma \in \Sigma$ is available exactly once in both colors. We call such instances *1-regular 2-colored* instances. For convenience, we restate this restricted version of the PPW.

Problem 5.1 1-REGULAR 2-COLORED PPW (1R2C-PPW)

Instance *An instance of the PPW with $|C| = 2$ and $|\Sigma| = \frac{n}{2}$, in which each letter of Σ occurs exactly once in both colors.*

Question *Find a permutation $\tau : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that $\tau(w) = w$ and the number of color changes within $\tau(c)$ is minimized.*

Recall that we restricted the PPW in the hope of finding polynomial time solution algorithms. We show in Section 5.2 that even the restriction of the PPW to the 1R2C-PPW is still $\mathcal{APX}$-hard. However, Section 5.5 describes the equivalence of the 1R2C-PPW to the problem of finding a shortest circuit in a certain class of binary matroids. Consequences of this equivalence are polynomial time solution algorithms for specific instances in this class and a duality theorem.

Example 5.1 *The word*

$$w = (A, B, B, C, D, A, C, E, E, D)$$

is a 1R2C-PPW instance with $\Sigma = \{A, B, C, D, E\}$ and $\gamma(w) = 5$.

5.1 Preliminary remarks

As in Chapter 4, we denote instances of the 1R2C-PPW by w. There is an insightful way (see [EHL03]) to represent instances and solutions of the 1R2C-PPW, which is shown in Figure 5.1.

Definition 5.1 *If the two occurrences of a letter $\sigma \in \Sigma$ appear at position x_σ and y_σ in a 1R2C-PPW instance w, we represent this letter pair with the interval $I_\sigma := [x_\sigma, y_\sigma]$. The set of all letter pair intervals of w is denoted by $I(w) := \{I_\sigma : \sigma \in \Sigma\}$.*

Figure 5.1: The interval representation and an optimal coloring with $\gamma^*(w) = 4$ for the 1R2C-PPW instance w of Example 5.1.

A solution of the 1R2C-PPW can thus be seen as a set $\{q_i\} \subseteq \{j + \frac{1}{2} : 1 \leq j \leq |w| - 1 \text{ and } j \in \mathbb{N}\}$ of rational numbers. Each q_i that intersects one or more intervals of $I(w)$ can be interpreted as a color change between the consecutive letters w_i and w_{i+1} of w. Observe that a solution for the 1R2C-PPW is feasible if and only if $|\{q_i\} \cap I_\sigma|$ is odd for all $I_\sigma \in I(w)$, i. e. there is an odd number of color changes within every interval I_σ.

There exist instances of the 1R2C-PPW that are easy to solve. Palindromes, for example, can always be colored with $\gamma^*(w) = 1$ color change. Lemma 5.1 gives another example.

Lemma 5.1 *If $I(w)$ is a clutter, then $\gamma^*(w)$ can be computed by the greedy algorithm and each interval $I_\sigma \in I(w)$ contains exactly one color change.*

Proof. The greedy strategy passes through w from the left to the right and applies a color change as late as possible. Suppose that $I_\sigma = [x_\sigma, y_\sigma]$ is the interval with minimal value of x_σ that contains at least three color changes. Then, according to the greedy strategy, the first resp. the second color change in I_σ is forced by an interval $I_\rho = [x_\rho, y_\rho]$ resp. $I_\tau = [x_\tau, y_\tau]$, and $y_\rho < y_\tau < y_\sigma$ holds. Moreover, as $I(w)$ is a clutter, $x_\rho < x_\sigma$ and $x_\tau < x_\sigma$ must hold. Now it follows that I_τ contains at least three color changes, which contradicts the minimality of x_σ. ■

However, Example 5.2 shows that the greedy algorithm fails in general.

Example 5.2 *Let w be a 1R2C-PPW instance with $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ and $|\Sigma|$ even and w given by*

$$w = (\sigma_1, \ldots, \sigma_{|\Sigma|/2}, \sigma_{|\Sigma|/2}, \ldots, \sigma_{|\Sigma|}, \sigma_{|\Sigma|}, \sigma_1, \sigma_{|\Sigma|/2+1}, \ldots, \sigma_{|\Sigma|/2-1}, \sigma_{|\Sigma|-1}).$$

The greedy algorithm colors w with $|\Sigma| = \frac{n}{2} = \mathcal{O}(n)$ color changes, while the minimal number of color changes is $\gamma^*(w) = 3$.

An instance w of the 1R2C-PPW is called *decomposable* if $w = (w_1, w_2)$, where w_1 and w_2 are both subsequences of w that are again instances of the 1R2C-PPW. In such cases the interval set $I(w)$ can be partitioned into the two interval sets $I(w_1)$ and $I(w_2)$ and the 1R2C-PPW can be solved for w_1 and w_2 independently. In the following, we consider only instances which are not decomposable.

Definition 5.2 *Let w be a 1R2C-PPW instance. The $(|\Sigma| \times (n-1))$-matrix $A(w) = (a_{\sigma j})$ is defined with respect to the interval set $I(w)$ of w by*

$$a_{\sigma j} := \begin{cases} 1 & , \text{ if } x_\sigma \leq j < y_\sigma \\ 0 & , \text{ otherwise} \end{cases}$$

for $\sigma \in \Sigma$.

This way every column of $A(w)$ corresponds to a possible position of a color change in w (see Figure 5.2). We may write A instead of $A(w)$ for short.

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Figure 5.2: An instance w of the 1R2C-PPW and the associated matrix $A(w)$.

5.2 Complexity results

The theorem presented in this section arose from collaboration with Paul S. Bonsma (see [BEH03]). It proves the $\mathcal{APX}$-hardness of the 1R2C-PPW using an L-reduction from the following problem.

Problem 5.2 CUBIC GRAPH VERTEX COVER (3GVC)

Instance *A cubic undirected graph $G = (V, E)$.*

Question *Find a vertex cover $S \subseteq V$ of minimal cardinality.*

In [AK00], the 3GVC problem is shown to be $\mathcal{APX}$-hard. First we show that undirected graphs with maximal degree 3 have an ordering of the vertices and an orientation of the edges satisfying the following criterion:

Criterion 5.1 *Let $G = (V, A)$ be a directed graph with a complete order $\prec$ on the vertices. We say that G and $\prec$ satisfy Criterion 5.1 if every vertex $v \in V$ has at most one in-neighbor u with $u \prec v$ and at most one in-neighbor w with $v \prec w$, and every vertex $v \in V$ with degree 2 has exactly one in-neighbor.*

Lemma 5.2 *Every undirected graph G with maximal degree 3 has an ordering of the vertices and an orientation of the edges that satisfies Criterion 5.1. This vertex order and orientation can be found in polynomial time.*

Proof. We can assume $G = (V, E)$ to be connected. We proceed by induction on $|V|$. The assertion is true when $|V| = 1$. We consider four cases.

1. G has a leaf. This case is trivial.

2. There is a degree 2 vertex v with neighbors u, w and $uw \notin E$. Now $G' = G - v + uw$ has an orientation and a vertex order as claimed. Assume wlog. $u \prec w$. Use this orientation and order for G, with v added in this order such that $u \prec v \prec w$, and the edges uv and vw oriented as a subdivision of the arc between u and w in G'. Now for u and w the situation in G is the same as in G' and v has one in-neighbor and degree 2, so this order and orientation satisfies Criterion 5.1.

3. There is a degree 2 vertex v with neighbors u, w and $uw \in E$. The graph $G' = G - v$ has an orientation and a vertex order as claimed. Assume (u, w) is oriented this way. Then u has at most one in-neighbor x in G'. If there is such an x and $x \prec u$, insert v after u in this order, otherwise insert v before u in this order. Using the orientation of G' and adding (w, v) and (v, u) gives an orientation for G. Now w has one in-neighbor. The vertex u has two in-neighbors if and only if $d(u) = 3$ and in this case one in-neighbor appears before u in the order and one after. Otherwise u has one in-neighbor. The vertex v has one in-neighbor and $d(v) = 2$. Therefore this order and orientation satisfies Criterion 5.1.

4. All vertices have degree 3. Choose a vertex v. Consider $G' = G - v$ and denote the neighbors of v by x, y and z. G' has an orientation and vertex order as required. In G', the vertices x, y, and z all have degree 2. Let u be the in-neighbor of x and w be the in-neighbor of y. Observe that wlog. we can assume that either $u \prec x$ and $w \prec y$ or $x \prec u$ and $y \prec w$. In the first case, insert v in the order after x and y, and in the second case insert v before x and y. Using the orientation of G' and adding (v, x), (v, y), and (z, v) gives an orientation for G. Now x (y) has two in-neighbors, one ordered before x (y) and one after, and z and v both have one in-neighbor, so this order and orientation satisfies Criterion 5.1.

This describes a polynomial time algorithm to find a vertex order and orientation satisfying Criterion 5.1 for every undirected graph with maximal degree 3. ■

Theorem 5.1 *The 1R2C-PPW is $\mathcal{APX}$-hard.*

Proof. Let G be an instance of the 3GVC. Find an orientation of the edges and an order $\prec$ on the vertices that satisfies Criterion 5.1 (see Lemma 5.2). The resulting directed graph is also be denoted by $G = (V, A)$ with $V = \{1, \ldots, n\}$ and $u < v \Leftrightarrow u \prec v$. We use G to construct an instance for the 1R2C-PPW. This instance consists of n blocks of letters, one block for every vertex. For every $i \in V$ we introduce letters A_i, B_i and C_i that only appear in block i. For every arc $a \in A$ we introduce letter E_a that appears once in both of the blocks corresponding to the end vertices. For every vertex pair $i, j \in V$ with $i < j$ we introduce letter D_{ij}, that appears once in block i and once in block j.

Observe that because G satisfies Criterion 5.1, we can find for every vertex v a complete order $\prec_A$ on the arcs incident with v such that the following holds.

- If $a = (u, v)$ and $u < v$, then $a \prec_A b$ for every other arc b incident with v.
- If $a = (u, v)$ and $v < u$, then $b \prec_A a$ for every other arc b incident with v.
- If $a \in A$ is incident with u and v and $b \in A$ is incident with v and w and $u < v < w$, then $a \prec_A b$.

If vertex v is incident with arcs a, b, and c and the order $a \prec_A b \prec_A c$ satisfies the above properties, we introduce the block

$$A_i D_{1i} D_{2i} \ldots D_{(i-1)i} E_a E_b E_c B_i B_i D_{i(i+1)} D_{i(i+2)} \ldots D_{in} A_i C_i C_i$$

for v. Now order the blocks left to right from 1 to n. Taken together this gives the word w for the 1R2C-PPW (see Figure 5.3 for an example). In Figure 5.3, $E_{(1,3)}$ is placed before $E_{(1,2)}$ in block 1. This is an arbitrary choice, both options give a valid construction. The same is true for the placement of $E_{(4,1)}$ and $E_{(4,3)}$ in block 4.

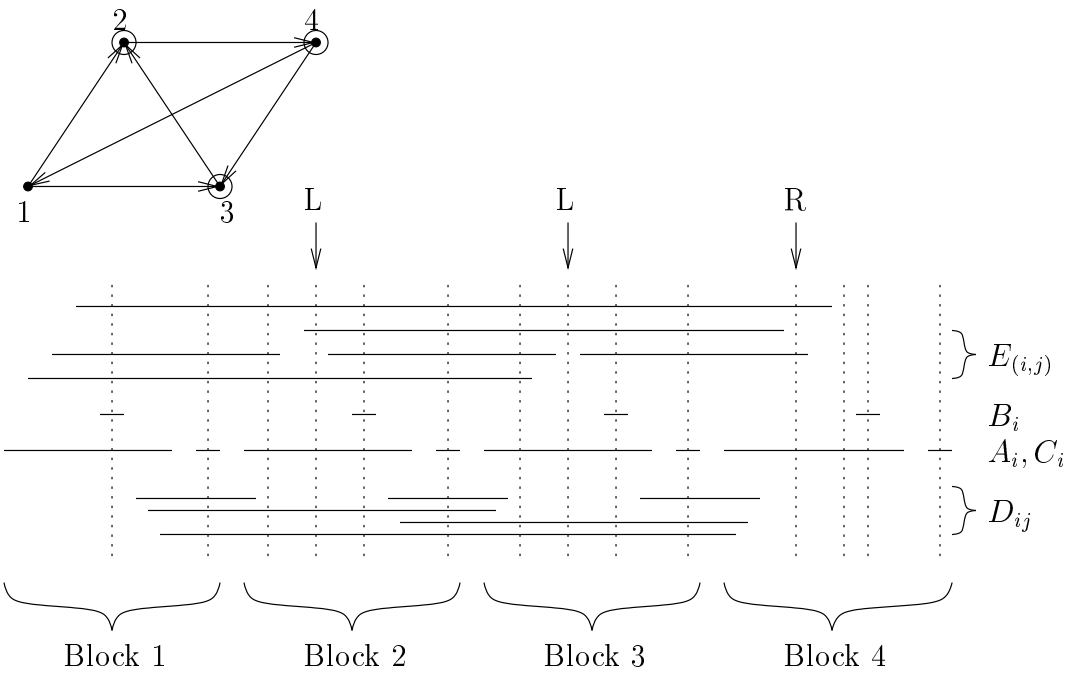


Figure 5.3: A 3GVC instance and solution, a solution for Criterion 5.1, and the corresponding 1R2C-PPW instance and solution.

Claim 5.1 *If the 1R2C-PPW instance w has a solution with at most $\gamma(w) = 2|V| + 2k$ color changes, then G has a vertex cover using at most k vertices.*

Proof. To prove this, we make the following observations that are true for every feasible solution for w.

1. Because of the B_i pair and the C_i pair in block i, there are at least two color changes in block i for every i.

2. If there is an arc $a \in A$ between i and j, then there are at least four color changes either in block i or in block j in any optimal solution for the 1R2C-PPW instance w.

We prove this last observation by contradiction. Suppose a is an arc between i and j, $i < j$ and in block i and in block j there are at most three color changes. Since one of the color changes in block i must be between C_iC_i, there are at most two color changes between the A_i pair. Since the number of color changes between a pair of letters must be odd, there is only one color change between the A_i pair. The same holds for the A_j pair. This color change between the A_i (A_j) pair must be between the B_i (B_j) pair. In block i, E_a appears before the B_i pair, whereas D_{ij} appears after it. In block j, E_a and D_{ij} both appear before the B_j pair. This means that there is an odd number of color changes between the E_a

pair if and only if there is an even number of color changes between the D_{ij} pair, a contradiction.

From the two observations above it follows that if there is a solution with at most $2|V| + 2k$ color changes, then G has a vertex cover with at most k vertices. ■

Claim 5.2 *If G has a vertex cover of cardinality k, there is a solution for the 1R2C-PPW instance w with $\gamma(w) = 2|V| + 2k$ color changes.*

Proof. Let S be a vertex cover of G. If $i \notin S$, we apply only two color changes in block i: One between B_iB_i and one between C_iC_i. If $i \in S$, we use four color changes in block i: Between B_iB_i and C_iC_i but also between the consecutive letters $D_{(i-1)i}E_a$ and between the consecutive letters E_cB_i. If there is an arc $a \in A$ with end vertices $i \in S$ and $j \in S$ $(i < j)$, we have to move one of the latter two color changes. If a is directed towards i, then E_aB_i is a consecutive letter combination in w and we move the color change one position to the left (between E_bE_a for some b). If a is directed towards j, then $D_{(j-1)j}E_a$ is a consecutive letter combination in w and we move the color change one position to the right (between E_aE_b for some b). This ensures that for every a between i and j with $i < j$, there is either one color change in block i between E_a and B_i and no color change in block j between $D_{(j-1)j}$ and E_a or there is no color change in block i between E_a and B_i and one color change in block j between $D_{(j-1)j}$ and E_a. In Figure 5.3, an example of such a 1R2C-PPW solution corresponding to a vertex cover is shown. The color changes marked with 'L' ('R') are the color changes moved to the left (right) in this last step. We prove that this method gives a feasible solution.

1. Let $i < j$. In block i, there is one color change to the right of D_{ij}, which is the color change between C_iC_i. In block j, there is no color change to the left of D_{ij}. In every block between i and j there is an even number of color changes. There are no color changes between blocks. So there is an odd number of color changes between the two D_{ij} letters for every $i < j$.

2. Let a be an arc between i and j, $i < j$. We know that either there is a color change in block i between E_a and the first B_i or that there is a color change in block j between D_{ij} and E_a, but not both. In addition, there is exactly one color change in block i between the first B_i and D_{ij} (this is the color change between B_iB_i). So there is an odd number of color changes between the E_a pair if there is an odd number of color changes between the D_{ij} pair. This was shown to be true, so for every a there is an odd number of color changes between the two E_a letters.

3. Between every A_i pair, there are either one or three color changes.

4. Between B_iB_i and between C_iC_i there is a color change.

We conclude that this is a feasible solution with exactly $2|V|+2|S|$ color changes. ■

Now suppose a $(1+\epsilon)$-approximation algorithm for the 1R2C-PPW exists. We use it to construct a $(1+3\epsilon)$-approximation algorithm for the 3GVC. Let $G = (V, E)$ be a 3GVC instance with minimal vertex cover S. We use the above transformation to construct a 1R2C-PPW instance w, with minimal number of $m := \gamma^*(w) = 2|S| + 2|V|$ color changes. Use the approximation algorithm to find a solution with $k \leq (1+\epsilon)m$ color changes. This gives a vertex cover of cardinality $k' \leq \frac{k}{2} - |V|$. Observe that since G is cubic, $|S| \geq \frac{1}{3}|E| = \frac{1}{2}|V|$, so $3|S| \geq |V| + |S|$ and

$$\frac{k' - |S|}{|S|} \leq \frac{k/2 - |V| - (m/2 - |V|)}{|S|} \leq 3\frac{k/2 - m/2}{|V| + |S|} = 3\frac{k-m}{m} \leq 3\epsilon$$

holds. So a PTAS for the 1R2C-PPW would give a PTAS for the 3GVC. Therefore, the 1R2C-PPW is also $\mathcal{APX}$-hard. ■

Corollary 5.1 *The 1R2C-PPW is $\mathcal{NP}$-complete.*

Proof. Use the same reduction as for the proof of Theorem 5.1. ■

5.3 Lower bounds

In addition to the upper bound given in Theorem 4.2, we describe an algorithm to derive a lower bound on the value $\gamma^*(w)$ for a 1R2C-PPW instance w. Observe that the maximal number of pairwise disjoint intervals in the interval set $I(w)$ corresponds to a maximal stable set in the interval graph of $I(w)$ and yields a straightforward lower bound on $\gamma^*(w)$. We generalize this idea to improve on the bound.

Example 5.3 *The interval set $I(w)$ for the 1R2C-PPW instance w in Figure 5.4 contains the maximal stable set $\{I_C, I_D, I_F, I_G\}$, which yields a lower bound of $\gamma^*(w) \geq 4$. As I_A must contain an odd number of color changes, the lower bound can be improved to $\gamma^*(w) \geq 5$. However, also I_B and I_E must contain an odd number of color changes. Consequently, the lower bound can be further improved to $\gamma^*(w) \geq 7$. In fact, $\gamma^*(w) = 7$.*

In the following, we consider the partial ordering on $I = I(w)$ given by proper containment and the corresponding directed acyclic graph $G_I^{\subsetneq} = (I, A)$, where $(I_\sigma, I_\tau) \in A :\Leftrightarrow I_\tau \subsetneq I_\sigma$. Example 5.3 suggests that the straightforward lower bound on $\gamma^*(w)$ can be improved by computing a maximum weighted stable set in the undirected subgraph induced by the out-neighbors of each vertex of $G_I^{\subsetneq}$ rather than computing only a maximal stable set in the interval graph of I.

Figure 5.4: Illustration of the lower bound computation.

Problem 5.3 MAXIMUM WEIGHTED STABLE SET PROBLEM (MWSSP)

Instance *A vertex-weighted undirected graph* $G = (V, E; w)$.

Question *Find a subset* $K \subseteq V$ *such that no two vertices* $i, j \in K$ *are adjacent in* E *and* $\sum_{i \in K} w(i)$ *is maximized.*

The MWSSP is $\mathcal{NP}$-complete in general (see [GJ79]). It can, however, be efficiently solved on interval graphs.

Theorem 5.2 (Hsiao and Chuan and Chang [HTC92]) *The MWSSP can be solved in linear time on interval graphs, if the sorted list of interval endpoints is given.* ■

The undirected subgraph of $G_I^{\subsetneq}$ induced by the out-neighbors of any vertex of $G_I^{\subsetneq}$ corresponds to a subset of I and, therefore, is an interval graph. We apply Algorithm 5.1, which passes through $G_I^{\subsetneq}$ in a bottom-up fashion. A lower bound on the minimal number of color changes for a vertex $I_\sigma \in I$ of $G_I^{\subsetneq}$ is given by the value of a maximum weighted stable set in the undirected subgraph of $G_I^{\subsetneq}$ induced by the out-neighbors of I_σ (we assume that this value is computed by the method `maxWSS(`I_σ`)`). Each I_σ must contain an odd number of color changes. A vertex $I_\sigma \in I$ is called *processed* if it is already weighted with a lower bound. It is called *processable* if all out-neighbors of I_σ are processed.

Algorithm 5.1 Computation of a lower bound on $\gamma^*(w)$.

Construct the directed graph $G_I^{\subsetneq} = (I, A)$
Assign a weight $w(I_\sigma) = 1$ to all vertices $I_\sigma \in I$ with no out-neighbor

while there exists an unprocessed vertex in G **do**
  Choose a processable vertex $I_\sigma \in I$ and set $w(I_\sigma) =$ `maxWSS(`I_σ`)`
  if $w(I_\sigma)$ is even **then**
    Set $w(I_\sigma) = w(I_\sigma) + 1$
Add an extra vertex $\lambda \notin I$ to I and arcs $\{(\lambda, I_\sigma) : I_\sigma \in I\}$ to A
return `maxWSS(`λ`)`

Theorem 5.3 *Algorithm 5.1 computes a lower bound on* $\gamma^*(w)$ *for a 1R2C-PPW instance* w *with a time complexity of* $\mathcal{O}(|A|)$, *where* A *is the arc set of* $G_I^{\subsetneq}$.

Proof. By induction it is apparent that $w(I_\sigma)$ is a lower bound on the number of color changes in I_σ for each $\sigma \in \Sigma$. In the computation of this lower bound, the subgraph of $G_I^{\subsetneq}$ induced by the out-neighbors of a vertex $I_\sigma \in I$ corresponds to a subset of I and, therefore, is an interval graph, so that Theorem 5.2 can be applied. As each out-neighbor of a vertex corresponds to an arc in $G_I^{\subsetneq}$ and each arc is considered exactly once, Algorithm 5.1 requires a time complexity of $\mathcal{O}(|A|)$. Note, however, that $|A| = \mathcal{O}(|\Sigma|^2)$ e. g. for palindromes. ■

5.4 Solution by integer programming

Note that we may try to solve the 1R2C-PPW by the straightforward integer program depicted in Figure 5.5, where the vector $b := (w(I_{\sigma_1}), \ldots, w(I_{\sigma_{|\Sigma|}}))^T$ contains the interval weights for a 1R2C-PPW instance as computed by Algorithm 5.1. However, the integer program does not always yield an optimal or even feasible solution for the 1R2C-PPW.

$$\begin{aligned} \vec{1}^T x &\rightarrow \text{min!} \\ Ax - 2Iy &= b \\ x_i &\in \{0,1\} \\ y_i &\in \mathbb{N}_{\geq 0} \end{aligned}$$

Figure 5.5: An integer programming solution approach for the 1R2C-PPW.

Example 5.4 *Figure 5.6(a) shows a 1R2C-PPW instance w for which Algorithm 5.1 yields the vector $b = (1,3,1,1,1)^T$. The corresponding integer program is, however, infeasible. The optimal value would be $\gamma^*(w) = 3$, which is reached for the vector $b^* = (1,3,1,3,1)^T$.*

Figure 5.6(b) shows a 1R2C-PPW instance w for which Algorithm 5.1 yields the vector $b = (1,1,3,1,1)^T$. The corresponding integer program yields the suboptimal value $\gamma(w) = 4$, whereas the optimal value would be $\gamma^*(w) = 3$, which is reached for the vector $b^* = (1,3,3,1,1)^T$.

(a) (b)

Figure 5.6: Examples where the integer program of Figure 5.5 fails.

Based on computational experiments like those in Example 5.4, we pose the following conjecture and open question.

Conjecture 5.1 *Given a 1R2C-PPW instance w and the corresponding vector b computed by Algorithm 5.1, there does always exist a vector $b' \in \{0,2\}^{|\Sigma|}$, so that the integer program in Figure 5.5 yields the optimal value $\gamma^*(w)$ if b is replaced by $b + b'$.*

Open question 5.1 *Can the vector b computed by Algorithm 5.1 be efficiently modified so that the integer program in Figure 5.5 does always yield a feasible solution for the corresponding 1R2C-PPW instance? Moreover, do such a modification algorithm and the correctness of Conjecture 5.1 imply the existence of a 2-approximation algorithm for the 1R2C-PPW?*

5.5 Problem reformulation

An alternative to the integer program depicted in Figure 5.5 is the solution of the 1R2C-PPW by an integer program with constraints over GF(2) as depicted in Figure 5.7(a).

$$\begin{array}{rcl} \vec{1}^T x & \rightarrow & \text{min!} \\ Ax & \equiv & \vec{1} \bmod 2 \\ x_i & \in & \{0,1\} \end{array}$$

(a)

$$\begin{array}{rcl} \sum_{i=1}^{n-1} x_i & \rightarrow & \text{min!} \\ (A,\vec{1})x & \equiv & \vec{0} \bmod 2 \\ x_n & = & 1 \\ x_i & \in & \{0,1\} \end{array}$$

(b)

Figure 5.7: Integer programming with constraints over GF(2).

We replace the constraint $Ax \equiv \vec{1} \bmod 2$ by the equivalent constraints $(A,\vec{1})x \equiv \vec{0} \bmod 2$ and $x_n = 1$ to obtain the integer program depicted in Figure 5.7(b). We turn to matroid theory and interpret each column of $(A,\vec{1})$ as an element of the binary vector matroid $M[A,\vec{1}]$. Then we are faced with the problem of finding a shortest circuit of $M[A,\vec{1}]$ that contains the element $\vec{1}$.

Problem 5.4 Shortest circuit problem in a binary matroid (SCPBM)

Instance *A binary matroid $M = (E,\mathcal{I})$ with a special element l and a distance function $d : E \setminus l \rightarrow \mathbb{N}_{\geq 0}$.*

Question *Find a circuit C of M containing l such that $\sum_{e \in C \setminus l} d(e)$ is minimal.*

It is well-known that the SCPBM is $\mathcal{NP}$-complete in general (see [Tru92]), but that it can be solved in polynomial time for specific instances (see Theorem 5.6). For the remainder of this chapter, we interpret instances of the 1R2C-PPW as instances of the SCPBM and apply matroid terminology. In this context, the word "regular" is used with another meaning as introduced in Chapter 4. However, there should occur no confusion between regular matroids and regular instances of the PPW.

Definition 5.3 *We denote the class of binary matroids made up by 1R2C-PPW instances by*

$$\Omega := \{M : M = M[A, \vec{1}] \text{ for a 1R2C-PPW instance } w\}.$$

Note that, by construction, each matrix A of a 1R2C-PPW instance has the consecutive ones property for rows and is therefore totally unimodular (in fact, no two sequences of consecutive ones in A even start or end in the same column). Therefore, $M[A, \vec{1}]$ is a one-point-extension of a regular matroid.

5.6 Good-natured instances

The solvability of an instance $M \in \Omega$ of the SCPBM is correlated to the existence of F_7 and F_7^* minors in M and can be described in terms of flows and cuts in matroids.

Definition 5.4 *Let $M = (E, \mathcal{I})$ be a matroid with a specific element $l \in E$ and a distance function $d : E \setminus l \to \mathbb{N}_{\geq 0}$. We call a set $S = \{C_1, \ldots, C_k\}$ of circuits of M a* flow *through l of value k, if both (a) $l \in (C_i \cap C_j)$ for all $C_i, C_j \in S$ and $C_i \neq C_j$, and (b) $|\{C_i : e \in C_i\}| \leq d(e)$ for all $e \in E \setminus l$ hold.*

A cut *through l of value k is a cocircuit D of M with $\sum_{e \in D \setminus l} d(e) = k$ and $l \in D$.*

Definition 5.4 can be used for a straightforward generalization of the well-known MaxFlow-MinCut theorem for graphs.

Definition 5.5 *A binary matroid M has the* MaxFlow-MinCut (MFMC) property *if, for all distance functions $d : E \setminus l \to \mathbb{N}_{\geq 0}$ and for all choices of l, the value of a maximum flow through l equals the value of a minimum cut through l.*

While the value of a maximum flow always equals the value of a minimum cut in graphs, there is a single obstacle that may prevent equality in matroids, which is the matroid F_7^*.

Theorem 5.4 (Seymour [Sey77]) *Let $M = (E, \mathcal{I})$ be a binary matroid with a specific element $l \in E$. Then M has no F_7^* minor that contains l if and only if the value of a maximum flow through l equals the value of a minimum cut through l for all distance functions $d : E \setminus l \to \mathbb{N}_{\geq 0}$.* ■

Any binary matroid M for which both M and M^* have the MFMC property does, according to Theorem 5.4, not contain an F_7 or F_7^* minor. It is easy to check that neither F_7 nor F_7^* can be represented by a matrix over $\mathbb{R}$. As every minor of a regular matroid again is regular, these matroids can not be minors of a regular matroid. In fact, F_7 and F_7^* are the only excluded minors.

Theorem 5.5 (Tutte [Tut58]) *A binary matroid is regular if and only if it has no F_7 or F_7^* minor.* ■

The recognition of an F_7^* minor in a binary matroid can be carried out in polynomial time by the use of matroid decomposition (see [Tru92]). It is therefore possible to decide if a given binary matroid has the MFMC property or if it is regular. The algorithm itself, however, is not practicable. A practical and useful implementation of a more general minor-checking algorithm is available as a part of the MACEK project (see [Hli]).

Theorem 5.6 (Truemper [Tru92]) *There is a polynomial algorithm for determining whether a binary matroid with a special element l has the MFMC property. Moreover, there are polynomial time algorithms for the computation of a minimum cut and the solution of the SCPBM in MFMC matroids.* ■

A consequence of Theorem 5.6 is that solvable instances of the SCPBM, and therefore the 1R2C-PPW, are those which do not contain both an F_7 and F_7^* minor.

Theorem 5.7 *The 1R2C-PPW is solvable in polynomial time for an instance $M \in \Omega$ if M is a regular matroid or an MFMC matroid or the dual of an MFMC matroid.*

Proof. By Theorem 5.5 every regular matroid is an MFMC matroid and the SCPBM can be solved for MFMC matroids by Theorem 5.6. If M is the dual of an MFMC matroid, then M^* is an MFMC matroid. As a minimum cut in M^* corresponds to a shortest circuit in M, the SCPBM can, again according to Theorem 5.6, be solved for M as well. ■

Table 5.1 contains enumeration results about the number of 1R2C-PPW instances which contain either none, exactly one F_7 or F_7^* minor, or both. Decomposable instances are ignored, and only one representative from each class of equivalent instances is counted. Note that the number of instances which contain both minors increases rapidly, while the number of instances which contain exactly one minor increases rather more slowly if the size of Σ increases. The reason for that is an increasing connectivity of the instances, such that the existence of one of both minors implies the existence of the other (see [Oxl92]).

	$\lvert\Sigma\rvert = 3$	$\lvert\Sigma\rvert = 4$	$\lvert\Sigma\rvert = 5$	$\lvert\Sigma\rvert = 6$	$\lvert\Sigma\rvert = 7$
No F_7 or F_7^* minor	8	44	328	2869	28145
F_7 minor only	—	1	6	50	310
F_7^* minor only	—	1	2	24	115
F_7 and F_7^* minor	—	1	53	1283	27234
Total	8	47	389	4226	55804

Table 5.1: Enumeration results for 1R2C-PPW instances.

5.6.1 Regular matroids

It is well-known that, for regular matroids, the SCPBM can be solved by linear programming (see [Tru92]). As we are interested in consequences of linear programming duality, we give a short description of the solution approach.

Suppose that we are given an instance $M \in \Omega$ of the SCPBM that is proven to be regular as indicated in Theorem 5.6. We may assume that M is already given in its standard representative form with a totally unimodular signing, i. e. the binary matroid $M[A, \vec{1}]$ is represented by (I, B, e) over $\mathbb{R}$. We may also assume that the vector e of (I, B, e) corresponds to the vector $\vec{1}$ of $(A, \vec{1})$.

The linear program that solves the SCPBM for M is given in Figure 5.8(a). The corresponding dual program is depicted in Figure 5.8(b). We may reformulate the constraints of the dual program to $-\vec{1} \leq u \leq \vec{1}$ and $-\vec{1}^T \leq u^T B \leq \vec{1}^T$. Thus, u weights (or chooses) the rows of $(I, -I, B, -B)$ with integers from $\{0, \pm 1\}$.

Theorem 5.8 *Let $M \in \Omega$ be a regular instance of the SCPBM. Then the linear program in Figure 5.8(a) yields an optimal solution for M and, therefore, for the corresponding 1R2C-PPW instance.*

Proof. As we assume (I, B, e) to be totally unimodular, the linear program has only binary integer solutions. Suppose that an optimal solution x^* uses columns c_1 and c_2 of $(I, -I, B, -B)$ with $c_1 = -c_2$. Then, as $c_1 + c_2 = 0$, both c_1 and c_2 are redundant, which contradicts the optimality of x^*. ■

$$\begin{array}{rcl} \vec{1}^T x & \rightarrow & \min! \\ (I, -I, B, -B)x & = & e \\ x_i & \geq & 0 \end{array} \qquad \begin{array}{rcl} e^T u & \rightarrow & \max! \\ u^T(I, -I, B, -B) & \leq & \vec{1}^T \end{array}$$

(a) (b)

Figure 5.8: The linear program and its dual for the solution of the SCPBM for regular matroids $M[A, \vec{1}]$.

Note that Theorem 5.8 confirms Theorem 4.2 for instances of the 1R2C-PPW.

Proof of Theorem 4.2. Assume that we are given a 1-regular instance w of the PPW with a color set of size $|C| \geq 2$. We generalize Definition 5.1 and Definition 5.2 and introduce an interval set $I(w) := \{I_\sigma^i : 1 \leq i \leq |C|-1 \text{ and } \sigma \in \Sigma\}$ of intervals $I_\sigma^i := [x_\sigma^i, y_\sigma^i]$, where x_σ^i denotes the i-th and y_σ^i denotes the $(i+1)$-th occurrence of σ in w. Additionally, we introduce a $((|C|-1) \times (n-1))$-matrix A^σ defined by

$$a_{ij}^\sigma := \begin{cases} 1 & \text{, if } x_\sigma^i \leq j < y_\sigma^i \\ 0 & \text{, otherwise} \end{cases}$$

for $\sigma \in \Sigma$. Based on $I(w)$, the matrix $A(w)$ is given by

$$A(w) = \begin{pmatrix} A^{\sigma_1} \\ \vdots \\ A^{\sigma_{|\Sigma|}} \end{pmatrix}$$

and, therefore, is a $(|\Sigma|(|C|-1) \times (n-1))$-matrix which has a rank of at most $|\Sigma|(|C|-1)$. ■

Corollary 5.2 *Let w be an instance of the 1R2C-PPW. Then $\gamma^*(w) \leq |\Sigma|$ holds.*

Proof. Let $N := \{i : e_i \neq 0\}$, where e is the right hand side vector of Figure 5.8(a). As both e and any optimal solution u^* of the dual program of Figure 5.8(b) contain only integers from $\{0, \pm 1\}$, it follows that $e^T u^* \leq |N| \leq |\Sigma|$. ■

5.7 MaxFlow-MinCut duality

In order to derive a duality result we need the following notion, which is the dual version of Definition 5.4.

Definition 5.6 *Let $M = (E, \mathcal{I})$ be a matroid with a specific element $l \in E$ and a distance function $d : E \setminus l \to \mathbb{N}_{\geq 0}$. We call a set $S = \{C_1, \ldots, C_k\}$ of cocircuits of M a* coflow *through l of value k, if both (a) $l \in (C_i \cap C_j)$ for all $C_i, C_j \in S$ and $C_i \neq C_j$, and (b) $|\{C_i : e \in C_i\}| \leq d(e)$ for all $e \in E \setminus l$ hold.*

Just as a flow can be interpreted as a disjoint packing of circuits, a coflow is a disjoint packing of cocircuits if we disregard the element l. In order to achieve a disjoint packing, each element $e \in E \setminus l$ with $d(e) > 1$ has to be replaced by $d(e)$ elements $e'_1, \ldots, e'_{d(e)}$ with $d(e'_i) = 1$ for each. We dualize Theorem 5.4.

Theorem 5.9 (Seymour [Sey77]) *Let $M = (E, \mathcal{I})$ be a binary matroid with a specific element $l \in E$. Then M has no F_7 minor that contains l if and only if the minimum length of a circuit through l equals the maximum value of a coflow through l for all distance functions $d : E \setminus l \to \mathbb{N}_{\geq 0}$.* ■

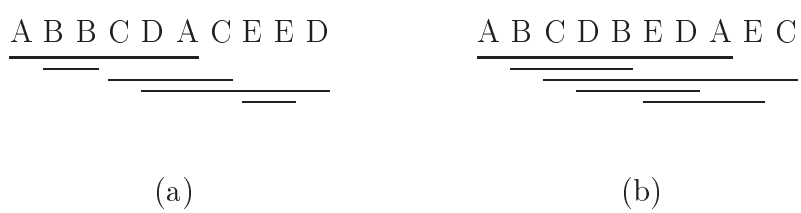
For any instance $M \in \Omega$, we have $M = M[A, \vec{1}]$, $l = \vec{1}$, and a distance function $d \equiv 1$. Any disjoint packing of cocircuits consists of GF(2)-sums (i. e. symmetric differences) of rows of A. These have to be odd, as each cocircuit must contain the element $\vec{1}$. We restate Theorem 5.9 in terms of the 1R2C-PPW, where the length of a shortest circuit that contains $\vec{1}$ corresponds to the minimal number of color changes for the corresponding 1R2C-PPW instance.

Theorem 5.10 *If $M[A, \vec{1}]$ has no F_7 minor that contains $\vec{1}$, the minimal number of color changes for a 1R2C-PPW instance equals the maximum value of a disjoint odd row sum packing of rows of A.*

Proof. This is a consequence of Theorem 5.9. ■

Example 5.5 *The 1R2C-PPW instance w in Figure 5.9(a) has a minimal value of $\gamma^*(w) = 4$, and $M[A, \vec{1}]$ contains no F_7-minor (in fact, $M[A, \vec{1}]$ is regular). A maximum disjoint odd row sum packing is given by $\{I_A \Delta I_B \Delta I_C\}$, $\{I_C \Delta I_D \Delta I_E\}$, $\{I_B\}$, and $\{I_E\}$.*

The 1R2C-PPW instance w in Figure 5.9(b) does contain an F_7-minor (to see this, contract the first column, add I_E to I_C, and contract the eighth column within $(A, \vec{1})$). It has a minimal value of $\gamma^*(w) = 3$, whereas a maximum disjoint odd row sum packing consists only of the two disjoint odd row sums $\{I_B\}$ and $\{I_E\}$. In fact, w contains an F_7^* minor as well (to see this, add I_E to I_C, contract the eighth column, add I_D to I_A, add I_B to I_D, add I_C to I_B and delete the seventh and ninth column within $(A, \vec{1})$).



(a) (b)

Figure 5.9: Instances of the 1R2C-PPW (a) without and (b) with an F_7 minor in $M[A, \vec{1}]$.

As mentioned in Section 5.6.1, the minimal number of color changes for a 1R2C-PPW instance can be computed by the linear program given in Figure 5.8(a) if the corresponding matroid $M \in \Omega$ is regular (see Example 5.6). We point out that an optimal solution of the dual program in Figure 5.8(b) does not, however, correspond to a maximum disjoint odd row sum packing in general, as it is suggested by Theorem 5.10.

Open question 5.2 *Given a 1R2C-PPW instance, how can a maximum disjoint odd row sum packing for the instance be efficiently computed? Does an algorithm have to distinguish between whether the instance yields a regular matroid, an MFMC matroid, or the dual of an MFMC matroid?*

Example 5.6 *The matroid $M[A, \vec{1}]$ associated to the 1R2C-PPW instance w depicted in Figure 5.10 is regular.*

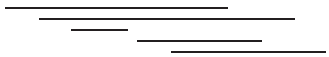


Figure 5.10: A 1R2C-PPW instance which yields a regular matroid $M[A, \vec{1}]$.

It can be represented over $\mathbb{R}$ by the matrix

$$(I, B, e) = \begin{array}{c} \begin{array}{ccccc|ccccc|c} 1 & 2 & 3 & 5 & 6 & 4 & 7 & 8 & 9 & 10 & \\ \hline \end{array} \\ \left(\begin{array}{ccccc|ccccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & I_A \Delta I_B \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & I_B \Delta I_C \Delta I_D \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & I_C \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & I_D \Delta I_E \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & I_E \end{array}\right) \end{array}$$

for which the rows sums and column permutations that occur as a result of the computation of the standard representative form of $(A, \vec{1})$ are given. The corresponding linear program given in Figure 5.8(a) has an optimal solution $x^* = (0, 0, 1, 0, 1|0, 0, 0, 0, 0|1, 0, 0, 0|0, 0, 0, 0)^T$, which chooses columns 3,6, and

4 of the original matrix A. Therefore, $\gamma^*(w) = 3$. *The dual program of Figure 5.8(b) yields an optimal solution* $u^* = (-1, 1, 1, 0, 1)^T$, *which does not give a maximum disjoint odd row sum packing of rows of A. In fact, a maximum disjoint odd row sum packing is given by* $\{I_A \Delta I_B \Delta I_E\}$, $\{I_B \Delta I_C \Delta I_D\}$, *and* $\{I_C\}$.

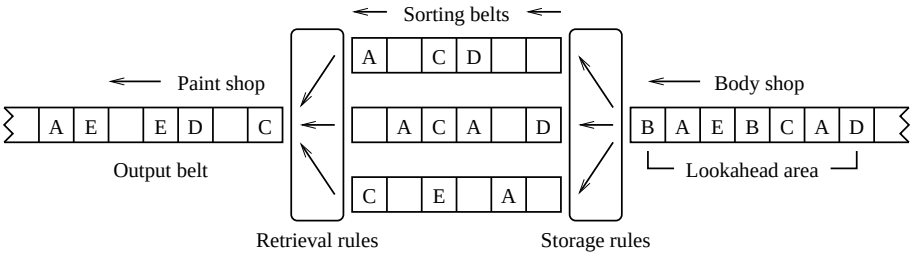
Part III

Algorithms

In contrast to the mainly theoretical results of Part II, the goal of this part is to find algorithms for color sequencing with respect to practical requirements. In particular, a given order sequence is allowed to be resorted by the use of an interim storage system. The aim of resorting is, again, the minimization of the number of color changes within the sequence that leaves the interim storage system. Thereby, we do not pay attention to the quality of the resorted order sequence with respect to any commodity other than the enamel color and, thus, identify an order with its designated enamel color. This can be justified in practice if there is another interim storage system placed after the paint shop that resorts the order sequence again, this time with respect to other commodities.

Interim storage systems are an essential tool for production control. They are installed in front of production shops to transform an incoming order sequence into an output sequence that increases the efficiency of the particular production shop and thus reduces production costs. Different types of interim storage systems exist and differ with respect to their flexibility and investment costs (see [WS95]). We consider the simplest, cheapest, and thus most commonly used type of interim storage systems, the line storage systems. Note that interim storage systems may not only be used for resorting, but also provide a safety stock for production.

A line storage system typically consists of a lookahead area L of length l that holds the next l elements of the input sequence, a set $Q_1, \ldots, Q_m$ of sorting belts on which the elements of the lookahead area are stored, each of length q, and an output belt on which the elements of the queues are retrieved. We take over a model presented in [Spi02] and represent the lookahead area by a stack, and each sorting belt by a queue.



The efficiency of a line storage system is clearly affected by its dimension, i. e. the values of l, q, and m. Additionally, production dependent factors like the number of colors that have to be processed, as well as the occupancy of the sorting belts have a significant impact. Typical values for the dimension of a line storage system that can be observed in practice are $l \in \{5, 10, 15\}$, $q \in \{6, 10\}$, and $m \in \{5, 6, 7, 10\}$. The cycle time available for the storage of a color on a queue or the retrieval of a color from a queue, respectively, ranges from 30 seconds up to several minutes. Thus, a color storage and retrieval strategy for a color sequence has to be computed online and in real-time (see [Spi02]).

We consider two offline problems that arise in the context of resorting with line storage systems. In Chapter 6, we solely focus on the retrieval of colors from the sorting belts, i. e. we ignore the lookahead area. Chapter 7 extends the first problem and additionally considers the storage of colors from the lookahead area on the sorting belts. Moreover, Chapter 8 describes a simulation model that is capable of the online and real-time computation of a color storage and retrieval strategy for color change minimization within arbitrary large production sequences.

A common way to deal with these problems is the use of heuristic rule sets (see [Spi02]). There exists, however, only little literature that deals with algorithms for the storage or retrieval of colors on or from line storage systems. (Note that [Gol80] describes the problem of resorting a sequence of integers by a line storage system.) We often cite [Spi02], who proposes several (exact and heuristic) solution approaches. We present new and exact solution algorithms for the color retrieval as well as for the color storage and retrieval problem, whereas our simulation model uses a greedy algorithm to combine several optimal partial solutions into a solution which is in general not optimal. We discuss computational results for each solution algorithm.

Chapter 6

The color retrieval problem

The problem of retrieving colors from a line storage system with a minimal number of color changes occurs as a subproblem of the production process whenever the line storage system has to be emptied (which, for example, may be necessary due to a breakdown of a production station that precedes the line storage system or due to maintenance). Additionally, the problem may occur as a subproblem within a more general context (see Chapter 7).

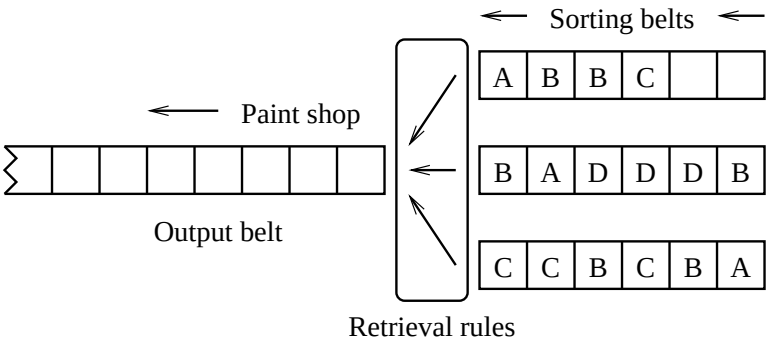


Figure 6.1: Illustration of the color retrieval problem.

Suppose that we are given a snapshot of a line storage system as depicted in Figure 6.1. As we focus on the retrieval of colors only, we interpret each sorting belt as a stack. The problem thus consists in the retrieval of the colors on the stacks such that the number of color changes within the retrieval sequence is minimized.

Problem 6.1 Color Retrieval Problem (CRP)

Instance *A set $S_1, \ldots, S_m$ of stacks of lengths $s_1, \ldots, s_m$ that contain colors of a color set C.*

Question *Retrieve the colors from $S_1, \ldots, S_m$ in a sequence R such that $\gamma(R)$ is minimized.*

For convenience, we assume in the following that R is initially empty and that the colors on each stack S_i occur consecutively. Note that, if R is supposed to start with a specific color $c_0 \in C$, we may achieve this by adding c_0 to the top of each S_i.

Example 6.1 *An optimal retrieval sequence with $\gamma(R) = 7$ color changes for the configuration depicted in Figure 6.1 would be*

$$R = (C_3C_3B_3C_3B_2B_3A_1A_2A_3D_2D_2D_2B_1B_1B_2C_1),$$

where the subscript of a color indicates the stack from which the color has been retrieved.

We have already mentioned that a common way to deal with the CRP in practice is the use of heuristic rule sets. In general, rules are made such that they (greedily) try to build as large color blocks within the retrieval sequence as possible. Exact solution approaches for the CRP include the formulation of the CRP as a sequential ordering or quadratic assignment problem as well as straightforward branch and bound methods (see [Spi02]). These approaches, however, do not provide an efficient way to solve the CRP.

We overcome this drawback and present an exact and efficient solution algorithm for the CRP by relating the CRP to the well-known multiple sequence alignment problem from computational biology. We start with a short sketch of multiple sequence alignment as it can be found in [Wat95].

6.1 Multiple sequence alignment

The alignment of multiple sequences is an important and frequent problem in computational biology, as it can be used to detect transformations of DNA or protein sequences.

Definition 6.1 *Let $S_1, \ldots, S_m$ be sequences over an alphabet Σ that does not contain the blank element $-$. An $(m \times n)$-matrix $A = (a_{ij})$ with $a_{ij} \in \Sigma \cup \{-\}$ is a* multiple sequence alignment (MSA) *of $S_1, \ldots, S_m$, if no column of A contains only blank elements and, ignoring the blank elements, row i of A equals S_i for $i = 1, \ldots, m$.*

Informally speaking, we obtain an MSA of $S_1, \ldots, S_m$ by the insertion of blank elements into $S_1, \ldots, S_m$, so that all sequences have the same length n afterwards. The alphabet Σ does usually consist of $|\Sigma| = 4$ (in case of the purine/pyrimidine

alphabet for DNA sequences) or $|\Sigma| = 20$ (in case of the amino acid alphabet of protein sequences) elements. However, the size of Σ is of no importance for the formulation of the MSA problem and a solution algorithm (note the observations in Section 6.3). Figure 6.2 shows an example of an MSA of four sequences.

```
GATTACA          -GATTAC-A
ATTAC            --ATTAC--
CAT              C-A-T----
TAG              ----TA-G-
```

Figure 6.2: Illustration of multiple sequence alignment.

Typically, a scoring function is used to rate the quality of an MSA, and an MSA with a minimal score value is said to be optimal. Among the different score schemes that may be applied, one of the most frequently used is the sum of pairs score.

Definition 6.2 *Let A be an MSA of $S_1, \ldots, S_m$. The* sum of pairs (SOP) score *of A is defined by*

$$D_d(A) := \sum_{1 \leq i < j \leq m} \sum_{k=1}^{n} d(a_{ik}, a_{jk}),$$

where $d : (\Sigma \cup \{-\})^2 \to \mathbb{N}_{\geq 0} \cup \{\infty\}$ is a distance function.

We delay the specification of a distance function d that fits our needs to Section 6.2. In the following, we deal with the SOP score solely. Given a set of sequences, the MSA problem thus consists in the computation of an MSA with a minimal SOP score.

Problem 6.2 Multiple sequence alignment (MSA) problem

Instance *A set of sequences $S_1, \ldots, S_m$ of lengths $s_1, \ldots, s_m$ over an alphabet Σ and a distance function $d : (\Sigma \cup \{-\})^2 \to \mathbb{N}_{\geq 0} \cup \{\infty\}$.*

Question *Compute an MSA of $S_1, \ldots, S_m$ with a minimal SOP score.*

The MSA problem can be shown to be $\mathcal{NP}$-complete by a reduction from the shortest common supersequence problem (see [GJ79]) and can be solved by dynamic programming.

Theorem 6.1 (Wang and Jiang [WJ94]) *The MSA problem is $\mathcal{NP}$-complete.* ■

6.1.1 Solution of the MSA problem

We illustrate the dynamic program for the solution of the MSA problem by a description of the alignment of two sequences

$$\begin{aligned} S_1 &= (a_{11}a_{12}\dots a_{1s_1}) \\ S_2 &= (a_{21}a_{22}\dots a_{2s_2}) \end{aligned}$$

with $a_{ij} \in \Sigma$.

The minimal value of an MSA of S_1 and S_2 can be computed be a dynamic program which iterates through all index pairs of S_1 and S_2, and checks whether it is best to align the two corresponding sequence elements or to align one of the sequence elements with the blank element. In other words, an optimal solution to the MSA problem is given by a shortest path from the vertex $s = (0, 0)$ to the vertex $t = (s_1, s_2)$ in an edge-weighted undirected graph $G = (V, E; w)$, where the vertex set

$$V = \{(v_1, v_2) : v_1 = 0, \dots, s_1 \text{ and } v_2 = 0, \dots, s_2\}$$

consists of all index pairs of S_1 and S_2, and the edge set

$$E = \bigcup_{e \in \{0,1\}^2 \setminus \vec{0}} \{(v, v + e) : v \in V \text{ and } (v + e) \in V\}$$

creates a grid, while the weight function w weights each edge $e \in E$ with its contribution to the SOP score (see [II94]).

The notation of the two-dimensional case can be straightforwardly extended to deal with the general case, in which we are given $m \geq 2$ sequences

$$\begin{aligned} S_1 &= (a_{11}a_{12}\dots a_{1s_1}) \\ S_2 &= (a_{21}a_{22}\dots a_{2s_2}) \\ &\vdots \\ S_m &= (a_{m1}a_{m2}\dots a_{ms_m}) \end{aligned}$$

with $a_{ij} \in \Sigma$. Algorithm 6.1 shows the dynamic program for the computation of an optimal MSA of m sequences, where $M_d(S_1, \dots, S_m)$ denotes the optimal value of an optimal MSA with respect to the distance function d and $d(\sigma_1, \sigma_2, \dots, \sigma_m) := \sum_{1 \leq i < j \leq m} d(\sigma_i, \sigma_j)$ denotes the contribution of the alignment of $\sigma_1, \sigma_2, \dots, \sigma_m$ to the SOP score.

Theorem 6.2 (Waterman [Wat95]) *Algorithm 6.1 solves the MSA problem with a space complexity of $\mathcal{O}(\Pi_{i=1}^m s_i)$ and a time complexity of $\mathcal{O}(2^m \Pi_{i=1}^m s_i)$.* ■

Algorithm 6.1 The dynamic program for the solution of the MSA problem.

Define $M_{i,j,\ldots,l} := M_d(a_{11}\ldots a_{1i}, a_{21}\ldots a_{2j}, \ldots, a_{m1}\ldots a_{ml})$
Set $M_{0,\ldots,0} = 0$ and $M_{0,\ldots,0,j,0,\ldots,0} = \sum_{k=1}^{j} d(-,\ldots,-,a_{jk},-,\ldots,-)$

for $i = 1$ to s_1 **do**
  for $j = 1$ to s_2 **do**
    for $\ldots$ **do**
      for $l = 1$ to s_m **do**
        Set $M_{i,j,\ldots,l} = \min_\epsilon\{M_{i-\epsilon_1,j-\epsilon_2,\ldots,l-\epsilon_m} + d(\epsilon_1 a_{1i}, \epsilon_2 a_{2j}, \ldots, \epsilon_m a_{ml})\}$,
        where the minimum is taken over all $\epsilon \in \{0,1\}^m \setminus \vec{0}$ and $0 \cdot a_{xy} := -$
return $M_{s_1,s_2,\ldots,s_m}$

Due to the rapidly increasing size of G, Algorithm 6.1 is suitable only for small instances of the MSA problem. A more sophisticated solution algorithm is the so-called A^* algorithm, which transforms edge weights before applying Dijkstras shortest path algorithm on G. The A^* algorithm is also known as best-first search, in contrast to breadth-first or depth-first search. Its name is justified by the fact that, at each search step, it uses a suitable heuristic lower bound estimator to focus on the most promising vertex $v \in V$, i. e. the vertex v for which the sum of the current shortest path length from the source s to v and the estimated shortest path length from v to the sink t is minimal (see [II94]).

There exist several implementations of the A^* algorithm. A very efficient one is described in [LR00], which uses not only a so-called dual feasible estimator for the computation of lower bounds, but also a dynamic heuristic upper bound and a clever way of iterating through all neighbors of a vertex.

6.2 MSA and the CRP

We show that the CRP can be modelled as an MSA problem. Recall that we are given a set $S_1, \ldots, S_m$ of stacks, each of which contains colors of a color set C. We identify each color $c \in C$ with an element of Σ (i. e. we set $C = \Sigma$) and equate each stack S_i with an input sequence for the MSA problem.

Our goal is to read off an optimal retrieval sequence R for the CRP from an optimal MSA matrix A by flattening A column by column into the sequence

$$R = (a_{11}a_{21}\ldots a_{m1}a_{12}a_{22}\ldots a_{m2}\ldots a_{1n}a_{2n}\ldots a_{mn}),$$

assuming that each stack is open to the left. Then, if we pass through R from the left to the right, we interpret an element $a_{ij} = c \in C$ as to retrieve color c from S_i and an element $a_{ij} = -$ as to retrieve nothing from S_i.

6.2.1 Preliminaries

We state first that we may assume that no stack S_i contains a color block $(c, \ldots, c)$ with $|(c, \ldots, c)| \geq 2$.

Proposition 6.1 *Given an instance of the CRP, the replacement of any color block $(c, \ldots, c)$ by the single color $c \in C$ on each stack does not increase the minimal number of color changes within the retrieval sequence for the instance.*

Proof. The minimal number of color changes within the retrieval sequence for the instance does clearly not increase if a representative for a color block is retrieved instead of the complete color block itself. ■

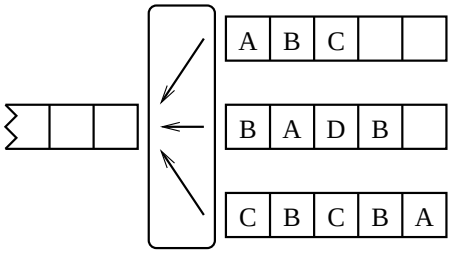


Figure 6.3: The CRP instance in Figure 6.1 after the appliance of Proposition 6.1.

After the appliance of Proposition 6.1 we solve the MSA problem for the modified stacks, which we denote for convenience again by $S_1, \ldots, S_m$. We complete Definition 6.2 by the specification of the distance function for the SOP score of an MSA.

Definition 6.3 *The distance function d for the SOP score of an MSA is given by*

$$d(x,y) := \begin{cases} 0 & , \text{ if } x = y \neq - \\ 1 & , \text{ if } x = - \text{ or } y = - \\ \infty & , \text{ otherwise} \end{cases}$$

for $x, y \in C \cup \{-\}$.

This (from a computational biology point of view highly unusual) definition of d forces any optimal MSA matrix A to possess specific properties.

Lemma 6.1 *Definition 6.3 prohibits that any column of an optimal MSA matrix A contains two or more different colors of C and forces A to contain as few columns as possible.*

Proof. Both properties follow immediately from Definition 6.3. ■

6.2.2 Equivalence of the MSA problem and the CRP

Now we are ready to prove that the solution of the CRP is equivalent to the solution of the MSA problem.

Lemma 6.2 *Let the solution of a CRP instance result in an optimal $(m \times n)$-MSA matrix A and a corresponding retrieval sequence R. Then R contains $\gamma(R) = n - 1$ color changes.*

On the other hand, let R be a retrieval sequence for the instance with $\gamma(R) = n-1$ color changes. Then we can construct a corresponding MSA matrix A with n columns and m rows, where m is the length of a longest color block in R.

Proof. The definition of the distance function d, the usage of the SOP score and Proposition 6.1 ensure that we get a color change in R if and only if we switch from one column of A to the next.

Conversely, for the construction of A, we pass through R from the left to the right, write each color block row by row in a different column, and assign the blank element $-$ to all yet undefined positions of A. ■

Finally, we prove that our solution approach solves an instance of the CRP to optimality.

Theorem 6.3 *Let A be an MSA matrix obtained after the solution of an instance of the CRP as described above. Then the flattening of A column by column gives an optimal retrieval sequence R for the instance.*

Proof. Suppose that R is not optimal and that there exists a retrieval sequence R' with $\gamma(R') < \gamma(R)$. According to Lemma 6.2, we can construct from R' a corresponding MSA matrix A' that contains less columns than A. Note that we can construct A' such that A' and A contain the same number of rows, as due to Proposition 6.1, the length of a longest color block in R' can not exceed the number of stacks. Then, according to Lemma 6.1, the fact that A' contains less columns than A contradicts the optimality of A. ■

Example 6.2 *Figure 6.4 shows an optimal MSA of the CRP instance in Figure 6.1, which yields the optimal retrieval sequence R given in Example 6.1.*

6.3 Computational results

Theorem 6.3 allows us to solve the CRP by the (ab)use of one of the several existing implementations of the A^* algorithm originally developed for the solution

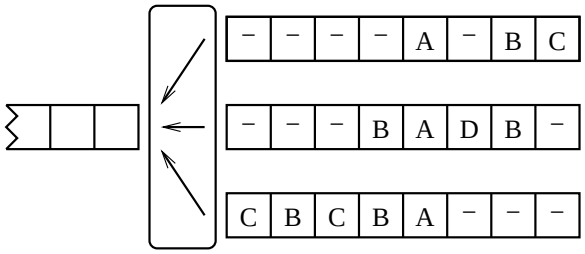


Figure 6.4: An optimal MSA for the CRP instance in Figure 6.1.

of the MSA problem. We use the implementation described in [LR00], which is an implementation by the authors. Although, as stated in [LR00], their use of the LEDA library (see [MN95]) imposes a significant space and time overhead, the obtained results do already attest the practical applicability of our solution approach to the CRP.

The parameter settings for our computational tests are given in Table 6.1 and refer to values that are observable in practice (see [Spi02]). The occupancy of the line storage system is fixed to 60% of its maximal possible occupancy, which is reached if every stack is filled up its maximal length s_i^*. This setting is due to a suggestion given in [Spi02].

Max. stack length s_i^*	No. of stacks m	No. of colors $\|C\|$
{6,10}	{5,6,7,10}	{5,10,15,20}

Table 6.1: Parameter settings for the CRP.

Our test instances are randomly created instances. We ensure that within each instance each color is used at least once and that $1 \leq s_i \leq s_i^*$ holds for each stack S_i. All running times are averaged over 10 instances, using a computer with 4 Sun UltraSparc II 400 Mhz processors and 3 GB memory.

No. of stacks	Max. stack length $s_i^* = 6$				Max. stack length $s_i^* = 10$			
	$\|C\| = 5$	$\|C\| = 10$	$\|C\| = 15$	$\|C\| = 20$	$\|C\| = 5$	$\|C\| = 10$	$\|C\| = 15$	$\|C\| = 20$
5	0.1	0.1	0.1	—[1]	0.1	0.1	0.1	0.1
6	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.2
7	0.1	0.1	0.1	0.2	0.2	0.3	0.2	0.3
10	0.5	1.0	1.1	0.7	8.4	13.4	23.5	18.9

Table 6.2: Running times in seconds for the solution of the CRP.

Our computational results are summarized in Table 6.2. The (averaged) running times are below 30 seconds for any parameter combination and for $m \neq 10$ stacks they are nearly negligible. The number of stacks has a significantly larger impact

[1] $5 \cdot 6 \cdot \frac{60}{100} = 18 < 20$.

on the running times than the length of the stacks (which is in correlation with Theorem 6.2) and maximal running times are reached for $m = 10$ stacks and $|C| = 15$ colors.

The running times for $m = 10$ in Table 6.2 seem to be influenced by the cardinality of the color set, although $|C|$ does not contribute to the complexity of the dynamic program for the solution of the MSA problem (see Theorem 6.2). In fact, an influence may arise sporadically due to our unusual definition of the distance function for the SOP score (see Definition 6.3) in conjunction with the bounding procedures used in the chosen implementation of the MSA algorithm ([REI]). Table 6.3 indicates, however, that it seems to be almost impossible to deduce any general effect of the cardinality of the color set on running times.

	Number of colors								
No. of stacks	6	7	8	9	10	11	12	13	14
10	0.9	0.6	1.9	1.1	1.0	0.4	2.2	0.5	0.3
11	2.8	6.2	10.8	4.5	4.9	10.3	3.7	4.0	5.2
12	34.1	10.5	19.9	25.9	14.8	31.6	75.7	37.7	18.6
13	69.4	90.8	304.8	80.6	532.8	98.0	57.3	119.9	123.2

Table 6.3: Standard deviations in seconds for the solution of the CRP with $s_i^* = 6$.

The standard deviations in Table 6.3 provide a closer look on instances with $m \in \{10, \ldots, 13\}$, broken down to a finer subdivision of the cardinality of the color set. They almost prove that not $|C|$, but rather the color distribution among the stacks influences the running time of the dynamic program for the solution of the MSA problem.

Chapter 7

The color storage and retrieval problem

This section extends the CRP to the simultaneous storage and retrieval of colors on resp. from a line storage system. The colors that have to be stored on the sorting belts are held in a lookahead area. The problem thus consists in the distribution of the colors that are held in the lookahead area among the sorting belts and the retrieval of all colors, so that the number of color changes within the retrieval sequence is minimized. Now, we model each sorting belt as a queue, and the lookahead area as a stack. Storage and retrieval operations are allowed to take place in any order.

Problem 7.1 Color storage and retrieval problem (CSRP)

Instance *A set $Q_1, \ldots, Q_m$ of queues, each having length q, and a stack L of length l that contain colors of a color set C.*

Question *Store the colors in L on $Q_1, \ldots, Q_m$, and retrieve all colors from $Q_1, \ldots, Q_m$ in a sequence R such that $\gamma(R)$ is minimized.*

Note that, in contrast to the CRP, we demand that all queues are of equal length q. This means that any queue that contains $q' < q$ colors must contain $q - q'$ gaps, which we represent by an additional color $c_0 \notin C$. Gaps within the retrieval sequence are ignored and they are not considered in the computation of the number of color changes for a sequence. We may assume (in analogy to the CRP) that R is initially empty and that the colors on each queue Q_i occur consecutively at the first q' positions of Q_i. The CSRP can, however, be formulated with queues of different length without major changes in the following sections.

Just like the CRP, the CSRP may be solved by an heuristic approach. Again, the retrieval rules could be made such that large color blocks appear within

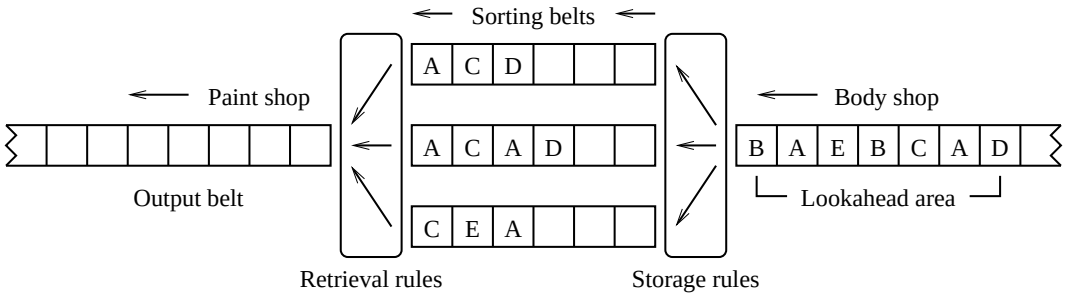


Figure 7.1: Illustration of the color storage and retrieval problem.

the retrieval sequence, whereas the storage rule could be made such that large color blocks appear on the queues. To our knowledge, the only exact solution algorithm for the CSRP consists in a straightforward branch and bound algorithm (see [Spi02]). In the following, we develop a dynamic program for the solution of the CSRP and show how its huge theoretical complexity can be drastically reduced in an implementation.

7.1 Solution by dynamic programming

We denote a state of our dynamic program by $S = (c; Q_1, \ldots, Q_m; p)$, where c denotes the last color on the retrieval sequence R of S, and $p \in \{1, \ldots, l\}$ denotes the current position in the lookahead area. We set $c = \emptyset$ if and only if $R = \emptyset$, and $p = l + 1$ to indicate that L is completely processed. We denote the color at lookahead position p by L_p. Any state $S = (c; \emptyset, \ldots, \emptyset; l + 1)$ is completely processed.

Given a state S, we assume that new states can be created by applying the following two basic operations to queues Q_i, Q_j of S.

(B1) Store the color L_p on Q_i.

(B2) Retrieve the first element of Q_j and move Q_j by one position.

Definition 7.1 *We denote the set of new states obtained after the appliance of B1 resp. B2 to queues Q_i, Q_j of a state S by $N^+(S)$ resp. $N^{\leftarrow}(S)$. Note that $|N^+(S)| \leq m$ and $|N^{\leftarrow}(S)| \leq m$ holds.*

The basic operations B1 and B2 may create gaps on queues and therefore within the retrieval sequence. Thus a color $c \in C$ or c_0 is retrieved whenever we move a queue. Recall that c_0 is ignored within the retrieval sequence.

Proposition 7.1 *Let $S = (c; Q_1, \ldots, Q_m; p)$ be a state.*

The set $N^+(S)$ is created as follows: Create a state

$$S' = (c; Q_1, \ldots, Q_{i-1}, Q_i', Q_{i+1}, \ldots, Q_m; p+1)$$

with $Q_i' = (c_1, \ldots, c_{q-1}, L_p)$ for every queue $Q_i = (c_1, \ldots, c_{q-1}, c_0)$ of S and add S' to $N^+(S)$.

The set $N^{\leftarrow}(S)$ is created as follows: Create a state

$$S' = (c_1; Q_1, \ldots, Q_{j-1}, Q_j', Q_{j+1}, \ldots, Q_m; p)$$

with $Q_j' = (c_2, \ldots, c_q, c_0)$ for every queue $Q_j = (c_1, \ldots, c_{q-1}, c_q)$ of S and add S' to $N^{\leftarrow}(S)$.

Proof. The correctness of this way of state creation follows from the basic operations B1 and B2. ■

Note that any empty queue $Q_i = \emptyset$ can be ignored for the creation of $N^{\leftarrow}(S)$, as it is unnecessary to move an empty queue. Moreover, in the creation of $N^+(S)$, we may push the color from the lookahead as far to the front of a queue as possible, as gaps do not affect the number of color changes for a state. We may thus assume that the last color of the retrieval sequence of a state is always a color $c \in C$. We denote the current number of color changes within the retrieval sequence R' of a state S' by $\gamma(S')$ and use Proposition 7.2 to compute $\gamma(S')$.

Proposition 7.2 *Let S' be a state. The value $\gamma(S')$ can be computed by*

$$\gamma(S') = \min_{P_1 \in P^+, P_2 \in P^{\leftarrow}} \{\gamma(P_1), \gamma(P_2) + \delta(P_2, S')\}$$

with $P^+ = \{P : S' \in N^+(P)\}$, $P^{\leftarrow} = \{P : S' \in N^{\leftarrow}(P)\}$, and δ defined by

$$\delta(P_2, S') := \begin{cases} 0 & , \text{ if } c = c' \text{ or } c = \emptyset \\ 1 & , \text{ otherwise} \end{cases}$$

for a state $P_2 = (c; Q_1, \ldots, Q_m; p)$ and $S' = (c'; Q_1', \ldots, Q_m'; p) \in N^{\leftarrow}(P_2)$.

Proof. The correctness of the computation of $\gamma(S')$ follows from the basic operations B1 and B2. ■

Thus, to compute $\gamma(S')$ for a state S', we have to consider $\gamma(S)$ of all states S from which S' can be created by the appliance of B1 or B2. In addition, if S' was created from S by the movement of a queue, we have to check whether an additional color change occurred in the retrieval sequence of S' or not. We formulate our dynamic program for the solution of the CSRP in Algorithm 7.1.

Algorithm 7.1 A dynamic program for the solution of the CSRP.

```
Create an initial state S_0 = (∅; Q_1, ..., Q_m; 1) with γ(S_0) = 0
Set P = {S_0}
while P ≠ ∅ do
  Set Q = ∅
  for all S ∈ P do
    for all S' ∈ N^+(S) ∪ N^←(S) do
      Compute γ(S') as described in Proposition 7.2
      if S' is not completely processed then
        Set Q = Q ∪ {S'}
      else
        Update best solution
  Set P = Q
return best solution
```

Theorem 7.1 *The dynamic program depicted in Algorithm 7.1 solves an instance of the CSRP requiring a time complexity of* $\mathcal{O}(|C|^{mq} \cdot |C| \cdot l \cdot m)$ *and a space complexity of* $\mathcal{O}(|C|^{mq} \cdot |C| \cdot l)$.

Proof. Our basic operations B1 and B2 are sufficiently elementary to admit the creation of any feasible state. For any feasible state S Proposition 7.2 ensures that the value $\gamma(S)$ is always the smallest possible.

For every lookahead position p we have $\mathcal{O}(|C|^{mq})$ possibilities to distribute the colors of the color set C on $Q_1, \ldots, Q_m$ (recall our treatment of gaps on the queues and the retrieval sequence of a state). Likewise, we have $\mathcal{O}(|C|)$ possibilities for the last color on the output belt (ignoring c_0). The computation of the minimal number of color changes for a state requires a complexity of $\mathcal{O}(m)$. ■

7.2 Implementation details

The huge theoretical complexity of Algorithm 7.1 can be drastically reduced in practice. Some reasons for that can be found in the avoidance of state duplicates or the avoidance of equivalent states that differ only by a permutation of their queue sets. We list and describe the main ingredients of our implementation.

7.2.1 Avoiding state duplicates

The appliance of B1 and B2 to a state S results in at most $2m$ new states (see Definition 7.1). However, as the sequence of basic operations that lead to a state

is not unique, some of the new states may already have been created before. A mechanism to avoid state duplicates can be applied during the computation of $\gamma(S')$ for a new state S' as described in Proposition 7.2. The computation of $\gamma(S')$ uses all states S for which $S' \in N^+(S) \cup N^{\leftarrow}(S)$ holds. By suitable bookkeeping, we can prevent that a duplicate of S' is created from a state S.

7.2.2 Removing equivalent states

We call two states $S_1 = (c; Q_1, \ldots, Q_m; p)$ and $S_2 = (c; Q'_1, \ldots, Q'_m; p)$ *equivalent*, if a permutation $\pi \in \mathcal{S}_m$ with $Q'_i = Q_{\pi(i)}$ for $i = 1, \ldots, m$ exists. Equivalent states can be easily identified if we keep the queues of a state in lexicographical order. If two states are equivalent, it is sufficient to keep only the state with the smaller number of color changes.

7.2.3 Expanding states

There are possibilities to promote the processing of a state $S = (c; Q_1, \ldots, Q_m; p)$.

If $c \neq \emptyset$ and there exists a queue Q_i that contains c at its front, we can retrieve c from Q_i (using the same argument as in Proposition 6.1). We may repeat this until every Q_i begins with a color different from c (or is empty).

If each queue has enough space to hold all remaining colors of the lookahead (from position p to l), we can store the color L_p on any queue that contains L_p as its last color (or, if such a queue does not exist, is empty) and set $p = p + 1$. We may repeat this until every Q_i has a last color different from L_p, or $p = l + 1$ holds.

7.2.4 Lower bounds

A lower bound on the minimal number of color changes within the retrieval sequence of a state S is given by the fragmentation of the colors on the queues (see [Spi02]).

Definition 7.2 *Let $Q_i = (c_{i1}, \ldots, c_{iq})$ be a queue of a state S. The* fragmentation $F(c, Q_i)$ of a color $c \in C$ on Q_i is defined by

$$F(c, Q_i) := |\{c_{ij} \in Q_i : c_{ij} = c \text{ and either } c_{ij} \neq c_{i,j+1} \neq c_0 \text{ or } c_{ij} = c_{iq} \neq c_0\}|.$$

As the fragments of a color on a queue cause unavoidable color changes within the retrieval sequence, we get a lower bound if we sum up the maximal fragmentation of each color.

Lemma 7.1 (Spieckermann [Spi02]) *The value*

$$B_l(S) := \sum_{c \in C} \max_{1 \leq i \leq m} \{F(c, Q_i) : Q_i \text{ is a queue of } S\}$$

is a lower bound on the minimal number of color changes for a state S. ■

Note that the lower bound can be increased by 1 for every color $c \in C$ for which $c \in L$, but $c \notin \bigcup_{i=1}^{m} Q_i$ holds.

7.2.5 Upper bounds

As soon as the lookahead of a state S is completely processed, the CSRP for S turns into a CRP for S, which can be solved as described in Chapter 6. Its solution yields an upper bound on the minimal number of color changes of an optimal state for the CSRP.

We prefer the solution of a CRP over the solution of a CSRP for such states due to reasons of complexity. The solution of the CRP requires (see Theorem 6.1) a time complexity of $\mathcal{O}(2^m \Pi_{i=1}^{m} |Q_i|)$, while Algorithm 7.1 requires a time complexity of $\mathcal{O}(m^{\sum_i |Q_i|})$.

Note that we may compute an upper bound at the start of Algorithm 7.1 by, for example, storing L completely on a queue Q_i and solving the resulting CRP. Any solution of a CRP may lead to an improvement of the upper bound. Any state S for which the value $\gamma(S) + B_l(S)$ exceeds the current upper bound can be discarded from further computation.

7.2.6 Hashing

Fast access to specific states is vital for the running time of Algorithm 7.1 and can be achieved by holding states in hash sets. Using integers instead of letters as color representatives, a hash function $h : S \to \mathbb{N}_{\geq 0}$ for a state S may be defined by

$$h(S) := \sum_{i=1}^{m} \sum_{j=1}^{q} i \cdot j \cdot c_{ij},$$

where $Q_i = (c_{i1}, \ldots, c_{iq})$ is a queue of S.

7.3 Computational results

Our implementation of Algorithm 7.1 is, despite of the improvements described in Section 7.2, not capable of solving realistic instances within an acceptable time limit.

The parameter settings for our computational tests are given in Table 7.1 and are chosen to illustrate the influence of each parameter on the running time of Algorithm 7.1. We point out that, in addition to the listed parameters, both the frequency of each color as well as the distribution of the colors at the lookahead and the queues influence the running time. Increasing color frequencies raise the effectiveness of the dynamic programming principle, while decreasing color frequencies are likely to result in an exponential number of states.

Queue length q	No. of queues m	No. of colors $\|C\|$	Lookahead length l
{5,6}	{5,6}	{5,6}	{5,6}

Table 7.1: Parameter settings for the CSRP.

Our test instances are built according to the principles described in Section 6.3. Again, all running times are averaged over 10 runs on a computer with 4 Sun UltraSparc II 400 Mhz processors and 3 GB memory.

Queue length	No. of queues	No. of colors	Lookahead length	Time
5	5	5	5	116
5	**5**	**5**	**6**	**519**
5	**5**	**6**	**5**	**434**
5	5	6	6	629
5	**6**	**5**	**5**	**794**
5	6	5	6	2440
5	6	6	5	1691
5	6	6	6	6093
6	**5**	**5**	**5**	**198**
6	5	5	6	381
6	5	6	5	771
6	5	6	6	1938
6	6	5	5	2362
6	6	5	6	10028
6	6	6	5	2450
6	6	6	6	7626

Table 7.2: Running times in seconds for the solution of the CSRP.

Table 7.2 summarizes our computational results. Bold lines indicate parameter settings with exactly one increased parameter. They indicate that the number

of queues has the most and the queue length has the least significant impact on the running times (although Theorem 7.1 suggests an equal importance of these parameters).

Chapter 8

CSRP-based simulation

The CSRP models the realities in a paint shop in a very general way. Therefore, it results neither in an applicable solution algorithm for the color change minimization nor does the problem formulation meet practical requirements. For example, recall that the CSRP allows an arbitrary order of color storage and retrieval operations. In practice, however, a constant frequency of storage and retrieval operations must be obeyed to avoid any interruption of the production flow.

This is a consequence of the most important practical restriction, the existence of a fixed production cycle time that has to be observed. Within each cycle, an unpainted car body enters the paint shop and a painted car body leaves the paint shop. Moreover, the fixed cycle time prescribes a maximal running time for the computation of an optimal storage and retrieval strategy for the colors currently in the lookahead area. Based on these observations, we introduce an altered version of the CSRP which is able to deal with practical requirements.

We imbed this altered version into a simulation model which is capable of a storage and retrieval simulation for a complete production sequence. Simulation is a common way to increase production productivity, as it has several advantages. On the one hand, the effect of applying different controls to a given production shop can be tested without any interruption of the production process. On the other hand, the effect of a modification of the production shop itself can be tested. While the first argument may result in a software change, the second may result in a hardware change. In either case, simulation is able to support decisions that increase production productivity.

8.1 A simulation model

Besides the existence of a fixed production cycle and the need of reasonable running times, an additional reason that prevents the simulation of a complete production sequence by the use of the CSRP only is the existence of manufacturing errors. The order sequence that is to enter the paint shop may change until shortly before the entry point. The small area that keeps the order sequence guaranteed not to be perturbed any more is the lookahead area of the line storage system.

In the following, we adapt several parts of the CSRP for an optimal storage and retrieval of the contents of the lookahead area, i. e. we compute an optimal storage and retrieval strategy for a subsequence of the complete production sequence only. The basic idea is to apply an adapted version of the CSRP to the second half of the lookahead area, while the first half is being processed with respect to the previously computed solution. To obtain a storage and retrieval strategy for the complete production sequence, we line up the strategies for all subsequences, hoping that this yields a reasonable solution.

The modifications of the basic operations of the CSRP are straightforward and yield only one feasible basic operation for CSRP-based simulation. We take over the notation from Chapter 7.

(B3) Store the color L_p on Q_i, retrieve the first element of Q_j and move Q_j by one position.

For convenience, we assume that the storage part of B3 takes place before the retrieval part.

Definition 8.1 *We denote the set of new states obtained after the appliance of B3 to queues Q_i, Q_j of a state S by $N_+^{\leftarrow}(S)$. Note that $|N_+^{\leftarrow}(S)| \leq m^2$ holds. As a consequence of B3, $N_+^{\leftarrow}(S)$ contains only states which have identical current lookahead area positions.*

In analogy to Proposition 7.1, we have the following.

Proposition 8.1 *Let $S = (c; Q_1, \ldots, Q_m; p)$ be a state.*

The set $N_+^{\leftarrow}(S)$ is created as follows: Create a state

$$S' = (c_{j1}; Q_1, \ldots, Q_i', \ldots, Q_j', \ldots, Q_m; p+1)$$

with $Q_i' = (c_{i1}, \ldots, c_{i,q-1}, L_p)$ *and* $Q_j' = (c_{j2}, \ldots, c_{jq}, c_0)$ *for every pair of queues* $Q_i = (c_{i1}, \ldots, c_{i,q-1}, c_0)$ *and* $Q_j = (c_{j1}, c_{j2}, \ldots, c_{j,q-1}, c_{jq})$ *of* S *and add* S' *to* $N_+^{\leftarrow}(S)$.

Proof. The correctness of this way of state creation follows from the basic operation B3. ■

Again, as in Section 7.1, we may push the color L_p from the lookahead area as far to the front of a queue Q_i as possible, so that again the last color of the retrieval sequence of a state is always a color $c \in C$.

Proposition 8.2 *Let* S' *be a state. The value* $\gamma(S')$ *can be computed by*

$$\gamma(S') = \min_{P \in P_+^{\leftarrow}} \{\gamma(P) + \delta(P, S')\}$$

with $P_+^{\leftarrow} = \{P : S' \in N_+^{\leftarrow}(P)\}$, *and* δ *defined by*

$$\delta(P, S') := \begin{cases} 0 & , \text{ if } c = c' \text{ or } c = \emptyset \\ 1 & , \text{ otherwise} \end{cases}$$

for a state $P = (c; Q_1, \ldots, Q_m; p)$ *and* $S' = (c'; Q_1', \ldots, Q_m'; p) \in N_+^{\leftarrow}(P)$.

Proof. The correctness of the computation of $\gamma(S')$ follows from the basic operation B3. ■

These modifications to the CSRP result in Algorithm 8.1, which is similar to Algorithm 7.1 and computes an optimal solution for the color storage and retrieval of the first half of the lookahead area with respect to B3.

Algorithm 8.1 A dynamic program for CSRP-based simulation.

```
Create an initial state S_0 = (∅; Q_1, ..., Q_m; 1) with γ(S_0) = 0
Set P = {S_0}
for p = 1, ..., l/2 do
  Set Q = ∅
  for all S ∈ P do
    for all S' ∈ N_+^←(S) do
      Compute γ(S') as described in Proposition 8.2
      Set Q = Q ∪ {S'}
  Set P = Q
return best solution
```

Lemma 8.1 *The dynamic program depicted in Algorithm 8.1 computes an optimal storage and retrieval strategy for the first $\frac{l}{2}$ colors in the lookahead area of a state S with respect to B3 requiring a time complexity of $\mathcal{O}((|C|+1)^{mq} \cdot |C| \cdot l \cdot m^2)$ and a space complexity of $\mathcal{O}((|C|+1)^{mq} \cdot |C| \cdot l)$.*

Proof. Proposition 8.2 ensures that the value $\gamma(S)$ is always the smallest possible for any state S which can be created by the appliance of B3. ■

We obtain a solution for a complete production sequence if we iterate through the complete sequence in steps of $\frac{l}{2}$ and line up all subsequence solutions computed by Algorithm 8.1. As our purpose is to point out the possibility of a color storage and retrieval simulation by means of the CSRP rather than to find out an optimal strategy for lining up subsequence solutions, we use a simple greedy strategy to obtain computational results, i. e. a best solution of Algorithm 8.1 is any solution $S \in P$ with a minimal value of $\gamma(S)$.

8.2 Implementation details and computational results

Most of the implementation details for the CSRP as described in Section 7.2 may also be used for CSRP-based simulation. An exception is the expansion of states, as the set $N_+^{\leftarrow}(S)$ in Algorithm 8.1 must contain only states which have identical current lookahead area positions.

Note that our simulation model offers an easy way to consider practical requirements (in this respect, it differs from the simulation model proposed in [Spi02]). In particular, it is possible to adapt the running time of our model to practical requirements, although this affects the number of enumerated solutions as well. We can guarantee the availability of a solution for the storage and retrieval of the second half of the lookahead area after $\frac{l}{2} \cdot t_c$ seconds, if we allow a running time of at most t_c seconds for the execution of a single basic operation B3 (meanwhile, the first half of the lookahead area is processed according to the previously computed solution). Additionally, we may add a restriction on the maximal number of states, and thus memory usage, or a maximal color block size on the output belt as it is done in [Spi02].

As in Section 6.3, the computational results are based on a line storage system whose occupancy is set to 60%. This means that colors are not retrieved from the line storage system until an occupancy of 60% is reached and the occupancy is kept constant afterwards. The cycle time is set to $t_c = 30$ seconds, and a memory restriction of 50.000 states is applied. Table 8.1 shows the parameter settings for

Queue length q	No. of queues m	Lookahead area length $\frac{l}{2}$
{6,10}	{5,6,7,10}	{3,5,7}

Table 8.1: Parameter settings for CSRP-based simulation.

the computational results, which are similar to the settings for the computational results for the CRP (see Table 6.1).

We use six randomly created color sequences for the computational results. Table 8.2 shows the color distribution within each sequence. The color distributions of S_1, S_2, S_3 are taken from [Spi02], while the color distributions of S_4, S_5, S_6 are taken from [PFO]. Each color sequence contains 2000 colors.

	Color									
Sequence	1	2	3	4	5	6	7	8	9	10
S_1	387	355	289	204	184	182	111	91	45	42
S_2	119	117	115	113	111	109	107	105	103	101
S_3	500	440	360	300	160	60	40	30	20	20
S_4	380	360	260	200	200	160	120	120	100	100
S_5	640	300	280	260	160	160	80	60	40	20
S_6	860	640	160	140	100	60	20	20	—	—

	Color									
Sequence	11	12	13	14	15	16	17	18	19	20
S_1	42	31	15	14	3	1	1	1	1	1
S_2	99	97	95	93	91	89	87	85	83	81
S_3	20	10	10	6	6	6	4	4	2	2
S_4	—	—	—	—	—	—	—	—	—	—
S_5	—	—	—	—	—	—	—	—	—	—
S_6	—	—	—	—	—	—	—	—	—	—

Table 8.2: Color distributions within the test sequences.

We rate the quality of the simulation results by the average size of color blocks in the output sequence of the line storage system. Table 8.3 shows the simulation results and gives the initial average color block size for each sequence on the input belt of the line storage system.

An apparent observation is that sequences that contain only few colors yield better improvements than sequences that contain many colors. While for $\frac{l}{2} = 3$ maximal color block sizes are obtained for $m = 10$ queues, for $\frac{l}{2} = 5$ resp. $\frac{l}{2} = 7$ maximal color block sizes are obtained for $m = 7$ resp. $m = 6$ queues. Note that the maximal color block size may decrease when the length of the lookahead area increases. Another counter-intuitive observation is that an increase of the queue length q does almost always lead to a decrease of the average color block

Sequence	Queue length $q = 6$				Queue length $q = 10$				Initial
	$m = 5$	$m = 6$	$m = 7$	$m = 10$	$m = 5$	$m = 6$	$m = 7$	$m = 10$	
S_1	1.9	2.0	2.1	2.3	1.9	2.0	2.1	2.3	1.2
S_2	1.5	1.6	1.6	1.8	1.5	1.6	1.6	1.7	1.1
S_3	2.1	2.3	2.4	2.6	2.1	2.3	2.3	2.6	1.2
S_4	1.9	2.1	2.3	2.6	1.9	2.1	2.2	2.6	1.2
S_5	2.1	2.3	2.6	2.8	2.2	2.3	2.5	2.8	1.2
S_6	2.8	3.3	3.4	4.1	2.9	3.1	3.4	3.9	1.4
S_1	2.1	2.2	2.2	2.1	2.0	2.1	2.1	2.0	1.2
S_2	1.6	1.7	1.7	1.6	1.6	1.7	1.7	1.6	1.1
S_3	2.3	2.5	2.6	2.4	2.2	2.4	2.5	2.2	1.2
S_4	2.1	2.3	2.4	2.2	2.1	2.2	2.3	2.1	1.2
S_5	2.4	2.5	2.6	2.5	2.3	2.5	2.5	2.3	1.2
S_6	3.2	3.5	3.6	3.3	3.1	3.3	3.5	3.1	1.4
S_1	2.0	2.1	2.1	1.8	2.0	2.1	2.0	1.7	1.2
S_2	1.6	1.6	1.6	1.5	1.6	1.6	1.6	1.4	1.1
S_3	2.3	2.5	2.4	2.1	2.3	2.4	2.3	2.0	1.2
S_4	2.1	2.2	2.1	1.9	2.1	2.2	2.1	1.8	1.2
S_5	2.3	2.5	2.4	2.1	2.3	2.4	2.4	2.0	1.2
S_6	3.3	3.4	3.2	2.7	3.0	3.1	3.2	2.6	1.4

Table 8.3: Average color block sizes for $\frac{l}{2} = 3$, $\frac{l}{2} = 5$, and $\frac{l}{2} = 7$.

sizes. This apparent contradictions can be explained by the chosen restriction on memory usage and cycle time, as the increase of any parameter is likely to cause a premature end of Algorithm 8.1 due to its increasing complexity.

The average color block sizes for the sequences S_1, S_2, S_3 illustrate the influence of the color frequencies within the sequences on the simulation results. Sequences that contain only few dominating colors (like S_3) yield better results than sequences in which all colors occur evenly (like S_2). The same observation applies to the sequences S_4 and S_5.

The improvement factors for the average color block sizes range from 1.5 to 3 and are inferior to those given in [Spi02]. The reason for that may be found in the simple greedy strategy used to line up the solutions of Algorithm 8.1. More sophisticated strategies are more likely to result in larger improvement factors (possible approaches can be found in [Spi02]). However, Table 8.3 shows that an increase of the average color block size can be achieved very easily, and that an adaption of the dynamic program for CSRP to practical requirements is possible. A definite reduction of at least 50% of the costs that arise in a paint shop seems to be achievable. The use of a line storage system in front of the paint shop is

therefore highly recommended.

Part IV

A real-world application

This part describes a real-world project that was successfully implemented in cooperation with the Ford Motor Company. It was aimed at reducing the number of color changes within the enamel booths of the companies automobile plants.

In contrast to Part II and Part III, where we modified an existing production sequence, we now have to compute a production sequence from scratch. Yet, different objectives for different production phases have to be taken into account. While the objective for the paint shop is, again, the reduction of the number of color changes, there exist objectives for the assembly shop as well which mainly consist of the support of a smooth production flow. We have already mentioned in Section 1.1 that savings due to an optimized production sequence are of most significance in the assembly shop.

A huge variety of criteria can be used to measure the quality of a production sequence and include, for example, utility work, labor utilization, and component usage (see [ZES97]). These criteria, however, usually focus on a single production shop only (mainly the assembly shop). We propose a model that covers the complete production process. Therefore, we identify the practical realities and obstacles that we have to cope with and describe the concepts we use to tackle with them. In particular, this includes an approach for the reduction of the number of color changes within the paint shop. Requirements like this and all other demands on the production sequence can be formulated by the use of a prioritized rule set and the observance of given rules is used as a measure for the quality of a production sequence.

Our two-stage solution approach also covers the occurrence of almost unavoidable manufacturing errors, which cause a delay of orders and therefore lead to a perturbation of the original production sequence (see Section 1.1). To our knowledge, both this anticipation of order delays and our approach for a reduction of color changes are not considered in the literature until now. We illustrate the efficiency of our solution approach, which is currently used for order sequencing in all automobile plants of the Ford Motor Company across Europe, by computational results on real-world data.

We would like to thank the Ford Motor Company, in particular Ralph Fischer, David Newton, Peter Oertel, and Simon Taylor, for a close and uncomplicated cooperation and for the permission to publish the following project description. However, for obvious reasons, we do not describe all details of our implementation.

Chapter 9

Order sequencing in the automobile industry

There exists a huge variety of order sequencing problems in practice and, therefore, just as much literature on this topic. However, each problem variant has its specific background that often prevents a generalization of ideas and algorithms, and successful solution approaches are often kept as company secrets (see [Ber01] for an overview over accessible reports).

Frequently, solution approaches to an automobile order sequencing problem are based on local search strategies (see [Hac00, PG02]). Other approaches employ constraint programming (see [Ber01]), while a classical approach is the well-known goal chasing algorithm (see [Mon98]). The basic idea, however, which all approaches have in common is the support of just-in-time production. The automobile industry is a leading user of this concept (see [Wil01]) that focuses on a smooth production flow. Although plants have a more complex structure, these approaches concentrate on a single part of the production process only, which is usually the assembly shop.

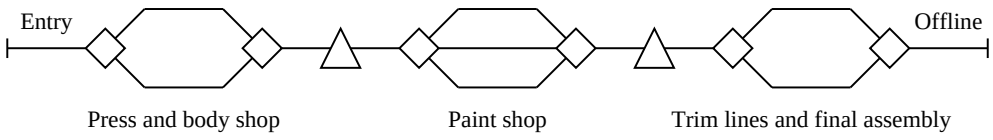


Figure 9.1: Sketch of an automobile production plant.

Figure 9.1 shows a sketch of a typical automobile production plant. Squares denote split points of the production sequence, while triangles denote interim storage systems that can be used for quality enhancement of the production sequence (see Part III) or production sequence recovery after manufacturing errors.

We assume that interim storage systems are used for production sequence recovery. Although this goal may not always be achieved completely, interim storage systems can provide a production sequence that is at least similar to the original production sequence. However, note that even a small perturbation of the original production sequence generally leads to a significant increase of the number of color changes in the paint shop. A focus on large subsequences of orders with identical enamel colors within the production sequence is therefore not sensible. Instead, we try to cope with avoidable perturbations of the original production sequence by enamel color clustering. The reason for that is the possibility of short-term enamel color interchanges during the production process as described in Section 1.1.

A variety of additional restrictions on the production sequence has to be respected. We formulate these restrictions in terms of rules and try to compute a production sequence that violates as few of these rules as possible. Note that, as the number of rules is usually large, it is in general not possible to arrange the orders in a production sequence that incurs no violation of rules at all.

In the following, we put this informal description in more concrete terms by a description of a model for the plant layout and the production process as well as a specification of the rule set and its evaluation. We describe our solution approach, which generalizes known solution approaches and illustrate its applicability.

9.1 Framework and basic concepts

The production process is essentially a probabilistic process. The main reason for that are lots of eventualities for manufacturing errors which make the production process difficult to control. We employ concepts that make the production process deterministic and thereby controllable. These concepts include a suitable representation of the plant, a sensible handling of manufacturing errors, and a suitable rule set for the evaluation of the quality of a production sequence. For convenience, Table 9.1 lists the meaning of the most frequently used abbreviations.

9.1.1 Orders, commodities, and combinations

We have already mentioned in Section 1.1 that each customer request is specified by a set of commodities.

Definition 9.1 *We denote the set of orders for a specific production day by $\mathcal{O}$ and the corresponding set of commodities by $\mathcal{C}$, and set $|\mathcal{O}| =: n$ and $|\mathcal{C}| =: m$.*

Notation	Meaning
$\mathcal{O}$	Order set of size n
$\mathcal{C}$	Commodity set of size m
$\mathcal{P}(C)$	Partition of the order set with respect to $C \subseteq \mathcal{C}$
$\mathcal{O}(C)$	Set of orders which contain each commodity in $C \subseteq \mathcal{C}$
$[o]$	Set of orders of the same commodity combination
$\mathcal{Z}$	Set of zones of the plant
$q(z)$	Quota (proportion of production) of a zone $z \in \mathcal{Z}$
z_m	Master zone of the plant
$R(i)$	Routed zones of master sequence position i
$P(o, i, z)$	Position of $o \in \mathcal{O}$ in $z \in \mathcal{Z}$ if sequenced to position i in z_m
S	Number of slots of the master zone, each of size $\frac{n}{S}$
$S^*(o)$	Set of preferred slots for an order $o \in \mathcal{O}$
$d(c, z)$	Delay of commodity $c \in \mathcal{C}$ in zone $z \in \mathcal{Z}$
$b(c, z_m; s)$	Banning factor for $c \in \mathcal{C}$ in $z_m \in \mathcal{Z}$ in slot s
$B(c, z_m; s)$	Banned positions for $c \in \mathcal{C}$ in $z_m \in \mathcal{Z}$ in slot s
$A(c, z_m; s)$	Desired average spreading distance for $c \in \mathcal{C}$ in $z_m \in \mathcal{Z}$ in slot s

Table 9.1: Abbreviations used in Chapter 9.

Each order $o \in \mathcal{O}$ can be interpreted as a vector $o \in \{0,1\}^m$, where

$$o_i := \begin{cases} 1 & \text{, if } o \text{ has commodity } i \\ 0 & \text{, otherwise} \end{cases}$$

for $i = 1, \ldots, m$. For convenience, we say that $c \in o$ or o *has commodity* c if $o_c = 1$.

The interpretation of orders as binary vectors assigns identical vectors to orders with identical commodity sets and yields a straightforward partition of $\mathcal{O}$.

Definition 9.2 *Let $a = (a_1, \ldots, a_m) \in \mathcal{O}$ and $b = (b_1, \ldots, b_m) \in \mathcal{O}$ be orders, and let $C \subseteq \mathcal{C}$ be a set of commodities. The resulting partition of $\mathcal{O}$ is denoted by*

$$\mathcal{P}(C) := [o]_1 \cup \ldots \cup [o]_p$$

for $p = 2^{|C|}$, where both $a \in [o]_i$ and $b \in [o]_i$ if and only if $a_c = b_c$ for all $c \in C$.

We call each set $[o]_i \subseteq \mathcal{P}(C)$ a commodity combination. *Thus, orders of the same commodity combination correspond with respect to all commodities of $\mathcal{C}$.*

9.1.2 Plant layout

We have already shown a typical plant layout in Figure 9.1. A plant consists of consecutive production shops and each production shop splits into several

lines which we call *zones*. Each zone, again, is split into *stations*. At least one commodity of an order is installed at each zone and exactly one commodity is applied at each station. Stations, however, are ignored in our model of the plant layout as the production process within a zone can not be influenced in any way.

Definition 9.3 *We denote the set of zones of the plant by $\mathcal{Z}$. The first zone of the plant is called* master zone. *We denote the master zone by $z_m \in \mathcal{Z}$ and call the order sequence in the master zone (which is the production sequence that has to be computed) the* master sequence. *Each position in the master sequence is an integer $p \in \{1, \ldots, n\}$.*

A suitable representation of a plant is given by a directed graph $G = (\mathcal{Z}, A)$ which is a directed path with a "multiplied" stable set of vertices (see [Gol80]). Each zone corresponds to a vertex, and arcs correspond to the production flow between zones (see Figure 9.2).

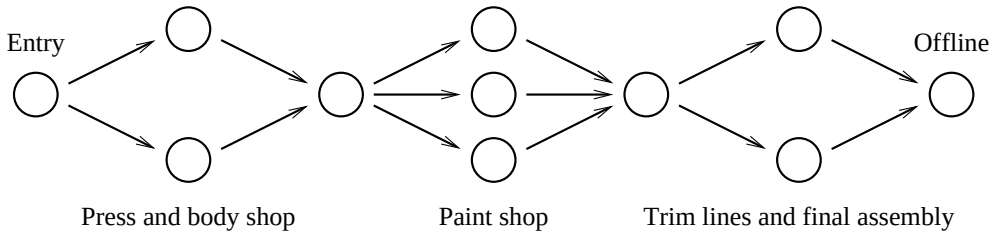


Figure 9.2: Representation of the plant of Figure 9.1 as a directed graph.

A zone $z \in \mathcal{Z}$ is called a *predecessor* of a zone z_0 if $z \neq z_0$ and there exists a directed path from z to z_0 in G. Likewise, z is called a *successor* of z_0 if $z \neq z_0$ and there exists a directed path from z_0 to z in G. We assume that $\mathcal{Z}$ contains exactly one zone without predecessor (the master zone) and exactly one zone without successor (the last zone of the plant). If two zones have the same in-neighbor, they are called *parallel*. Parallel zones must have the same out-neighbor. A zone which is not parallel is called *non-parallel*. If the in-neighbor of a zone $z \in \mathcal{Z}$ has more than one out-neighbor, z must have exactly one out-neighbor, i. e. a parallel zone must not be followed by another parallel zone. In contrast, a non-parallel zone may be followed by either a parallel zone or another non-parallel zone.

Parallel zones of a plant may differ in their length and speed. Therefore, each zone may have another proportion of production, which we call the *quota* of the zone.

Definition 9.4 *We denote the* quota *of a zone $z \in \mathcal{Z}$ by $q(z) \in \mathbb{Q}_{>0}$ and extend our plant representation to a vertex-weighted directed graph $G = (\mathcal{Z}, A; w)$, where*

$$w(z) := \begin{cases} 1 & , \text{ if } z \text{ is a non-parallel zone} \\ q(z) < 1 & , \text{ otherwise} \end{cases}$$

for $z \in \mathcal{Z}$.

We assume that zone quotas are scaled so that $\sum_{z \in Z} q(z) = 1$ holds for any maximal set Z of parallel zones.

9.1.3 Commodity delays

We have already mentioned that the production process is essentially a probabilistic process. There is, however, a correlation between commodities and manufacturing errors, i. e. some commodities are more susceptible to manufacturing errors than others. Each manufacturing error leads to a production rerun that delays an order within the production sequence with respect to its original position. Therefore, we model (and anticipate) the occurrence of manufacturing errors in a deterministic way and introduce *commodity delays*. A commodity delay can be imposed on each zone of the plant.

Definition 9.5 *A* commodity delay *is a mapping* $d : \mathcal{C} \times \mathcal{Z} \to \mathbb{N}_{\geq 0}$, *and the delay of a commodity* $c \in \mathcal{C}$ *in a zone* $z \in \mathcal{Z}$ *is denoted by* $d(c, z)$.

Note that commodity delays may cause gaps in the order sequence of zones, which can be represented by dummy orders that act as wildcards and can, for example, be filled by orders of preceding or succeeding production days.

To our knowledge, there does not exist any other solution approach for an order sequencing problem that employs commodity delays.

9.1.4 Storage and retrieval patterns

Zone quotas of parallel zones are used for the generation of deterministic patterns for the storage and retrieval of orders in the zones. Orders are stored on a parallel zone with respect to this pattern from the preceding non-parallel zone. Likewise, orders are retrieved from the parallel zone to the succeeding non-parallel zone by the use of the same pattern.

Given a maximal set Z of parallel zones, a *pattern* is a sequence of integers $p_i \geq 1$, where each entry p_i is interpreted as the storage resp. retrieval of p_i orders on resp. from zone $i \bmod |Z|$. Consequently, we have $\sum_i p_i = n$ for each pattern.

Example 9.1 *Let* $Z = \{z_1, z_2, z_3\}$ *be a maximal set of parallel zones. Both the pattern* $P_1 = (1, 1, 1, 1, 1, 2, \ldots)$ *and the pattern* $P_2 = (2, 2, 3, 2, 2, 3, \ldots)$ *describe*

the storage and retrieval of orders for zone quotas $q(z_1) = q(z_2) = \frac{2}{7}$ *and* $q(z_3) = \frac{3}{7}$*. The proportion of production is therefore* $\frac{2n}{7}$ *for both* z_1 *and* z_2*, and* $\frac{3n}{7}$ *for* z_3*. In practice,* P_1 *is preferred over* P_2 *as it distributes orders more evenly on the zones and supports a smooth production flow in a better way.*

Note that the order sequence stored in a parallel zone equals the order sequence retrieved from a parallel zone, unless a commodity delay occurs.

9.1.5 Sequence index routing

Zones and patterns as introduced in Section 9.1.2 and Section 9.1.4 are used to compute a routing of master sequence order positions through the plant. The routing assigns to each order position i of the master sequence a set of zones through which an order passes if sequenced to position i. Likewise, the routing assigns to each zone z of the plant a set of order positions that pass through z. Note that this yields either a permutation (for non-parallel zones) or a partition into sequences (for parallel zones) of the order positions in the master sequence.

Definition 9.6 *A* routing *is a mapping* $R : \{1, \ldots, n\} \to 2^{\mathcal{Z}}$*, where* $R(i) \subseteq \mathcal{Z}$ *denotes the set of zones through which an order passes if sequenced to position* i *of the master sequence, and* $R^{-1}(z) := \{i \in \{1, \ldots, n\} : z \in R(i)\}$ *denotes the set of master sequence order positions that pass through a zone* $z \in \mathcal{Z}$*.*

We assume that the set $R(i)$ is not affected by commodity delays, i. e. commodity delays do not change the set of zones through which an order passes. Then the delay of an order (induced by the delay of its commodities) in a zone can be calculated as the sum of its maximal commodity delays in preceding zones. Note that zone quotas have to be respected.

Definition 9.7 *Let* $o \in \mathcal{O}$ *be sequenced to position* i *of the master sequence. The position of* o *in any zone* z_0 *succeeding* z_m *is given by*

$$P(o, i, z_0) := P(i, z_0) + \sum_{z \in Z_p} \max_{c \in o} \{d(c, z)\} \cdot q(z),$$

where $P(i, z_0)$ *denotes the position of the master sequence position* i *in zone* z_0 *as given by the routing, and* $Z_p \subseteq R(i)$ *denotes the set of zones preceding* z_0*.*

9.1.6 Sequence slots

The master zone and, with respect to zone quotas, all other zones of the plant are divided into *slots* of equal size, i. e. position i of the master sequence is contained in slot s if and only if position i is contained in slot s of any zone $z \in R(i)$. Note, however, that sufficiently large commodity delays may cause an order sequenced to slot s of the master zone to appear in a slot $s' > s$ of a succeeding zone.

Definition 9.8 *We denote the number of slots of the master zone by S and assume that the size $\frac{n}{S}$ of each slot is an integer.*

The introduction of slots is one of the key concepts for enamel color clustering, as enamel colors can be advised to appear in a specified number of slots only. Furthermore, the master sequence is computed slot by slot which significantly decreases the running time of our solution algorithm. In practice, each slot usually corresponds to a working shift.

9.1.7 Rules

Each zone of the plant accepts rules. Each rule must state the zone and the commodity it applies to and a priority. The priority of a rule is an integer $p \in \{1, \ldots, 10\}$, where $p = 1$ denotes the highest and $p = 10$ the lowest priority.

Each rule specifies a constraint on a zone of the plant and the quality of a sequencing of an order o to a position i of the master sequence is rated by the number of resulting rule breaks. Note that, as we use a deterministic routing of order positions through the plant, it is possible to evaluate each rule in the master zone instead of the zone it applies to. The following rules are accepted.

Banning rule Commodities may be banned from parts of zones. Therefore, it is possible to specify at most one interval, i. e. a set $B(c, z) \subseteq \{1, \ldots, |z|\}$ of consecutive integers for any commodity $c \in \mathcal{C}$ and any zone $z \in \mathcal{Z}$, meaning that an order with commodity c should not occur at any position $i \in B(c, z)$ in zone z (here, $|z|$ denotes the length of z). Using the deterministic sequence index routing described in Section 9.1.5, we can deduce banned order sequence positions for each slot of the master zone.

Definition 9.9 *We denote the set of banned commodity sequence positions for slot s of the master zone by $B(c, z_m; s)$. The corresponding* banning factor *is given by*

$$b(c, z_m; s) := \frac{n - S \cdot |B(c, z_m; s)|}{n - |B(c, z_m)|}$$

for $s = 1, \ldots, S$ and indicates the banning proportion for each slot s of the master zone z_m.

Note that $B(c, z_m) = \bigcup_{s=1}^{S} B(c, z_m; s)$ holds. If $B(c, z_m) = \emptyset$ or each slot s has the same number $|B(c, z_m; s)|$ of banned order sequence positions, we have $b(c, z_m; s) = 1$. Otherwise, we have $0 \leq b(c, z_m; s) < 1$ if the banning in slot s is below average and $b(c, z_m; s) > 1$ if the banning in slot s is above average. In any case, $\sum_{s=1}^{S} b(c, z_m; s) = S$ holds.

Clustering rule For each commodity $c \in \mathcal{C}$ can be specified by an integer $1 \leq S(c, z) \leq S$ that c should occur in at most $S(c, z)$ slots in zone $z \in \mathcal{Z}$. However, if $C \subseteq \mathcal{C}$ is the set of commodities which are affected by a clustering rule, any order $o \in \mathcal{O}$ is allowed to have at most one commodity $c \in C$. Furthermore, if z is a parallel zone for which a clustering rule holds, we assume that the clustering rule holds for all zones which are parallel to z. Although this rule is intended to support enamel color clustering only, it can in principle be applied to other commodities, too.

Grouping rule Integers $g_{\min}(c, z) \geq 2$ and $g_{\max}(c, z) \geq g_{\min}(c, z)$ can be used to specify that a commodity $c \in \mathcal{C}$ should occur in zone z in a consecutive subsequence which has a minimal length $g_{\min}(c, z)$ and a maximal length $g_{\max}(c, z)$. Moreover, it is possible to specify a minimal distance $d(c, z) \geq 1$ between consecutive subsequences of orders that contain c.

Ratio rule The maximal number of occurrences of a commodity $c \in \mathcal{C}$ within a consecutive subsequence of the production sequence can be specified by a rational number $R(c, z) = \frac{x}{y} \in \mathbb{Q}_{\geq 0}$ which means that at most x occurrences of c are allowed within any consecutive subsequence of length y of the order sequence in zone z. Note that $R(c, z) = \frac{x}{y}$ and $R(c, z) = \frac{kx}{ky}$ for $k \in \mathbb{N}_{>0}$ do not formulate identical constraints in general.

Spacing rule A minimal distance between orders with a commodity $c \in \mathcal{C}$ can be specified by an integer $D(c, z) \geq 1$ which means that between any two orders in the order sequence in zone $z \in \mathcal{Z}$ that have commodity c there should be at least $D(c, z)$ orders that do not have commodity c.

Spreading rule An even spreading of a commodity $c \in \mathcal{C}$ within the order sequence of a zone $z \in \mathcal{Z}$ can be demanded. Evenly spread commodities are essential for a smooth production flow.

Most of these rules are typical for automobile production. While they can be used to support a smooth production flow and just-in-time production, they provide opportunities to respect particular technical restrictions of plants as well. To our knowledge, the clustering rule is a new supplement in this context.

We may assume that the commodity set $\mathcal{C}$ contains only commodities for which at least one rule holds. Note that there is no emphasis of any of the rules to be an objective function, i. e. a high-quality order sequence is any order sequence that incurs a minimal violation of these rules with respect to their priorities.

9.2 Solution approach

We use a two-stage approach for the computation of a master sequence.

1. Compute an order clustering that supports the subsequent computation of a master sequence slot by slot (described in Section 9.3).
2. Compute a master sequence that violates as few rules as possible (described in Section 9.4).

We remark that several other solution approaches employ local search algorithms for the improvement of a master sequence (see [Hac00] for an approach that uses a genetic algorithm or [PG02] for an approach that uses threshold accepting). The second stage of our approach does already suggest a way to rate different master sequences, which is necessary for the employment of a local search algorithm and described in more detail in Section 9.4.1. Whenever an order is placed on a position of the master sequence, it causes a certain set of rules to be broken. The accumulation of these rule breaks along the master sequence (with respect to rule priorities) can be used to rate its quality. We experimented with different local search strategies that use such a rating, namely different order exchange strategies in a simulated annealing context, which in fact lead to an improvement of a randomly computed master sequence. There has, however, been no improvement if the master sequence was computed as described in Section 9.3 and Section 9.4. This indicates the stability of our solution.

9.3 Order clustering

The master sequence is computed slot by slot to decrease the running time of our algorithm. We therefore have to assign each order to exactly one slot of the master sequence, which is done in an order clustering step that has to respect clustering rules. Additionally, we have to focus on an even distribution of non-clustering commodities among the slots, so that the orders of each slot can be treated as a smaller copy of the original order set.

The order clustering step is divided into two parts.

1.1 Compute a set of preferred slots of the master zone for each order (described in Section 9.3.1).

1.2 Assign each order to a slot of the master zone (described in Section 9.3.2).

Rules which basically can prevent an even spreading of commodities among the slots of the master zone are clustering and banning rules. We therefore consider banning, clustering, and spreading rules during the order clustering step.

Definition 9.10 *We denote the set of commodities affected by a banning, clustering, spreading rule by $\mathcal{C}_{Ban}, \mathcal{C}_{Clu}, \mathcal{C}_{Spr}$, respectively. Moreover, we denote the set of orders that have commodities from a specific commodity set by*

$$\mathcal{O}(C) := \{o \in \mathcal{O} : c \in o \text{ for all } c \in C\}$$

for $C \subseteq \mathcal{C}$.

9.3.1 Computation of preferred order slots

Our aim is to support an even spreading of commodities with respect to clustering and banning rules. We therefore try to distribute weighted proportions of commodity combinations representing these rules on each slot in a suitable way, i. e. the sum of weighted proportions assigned to each slot should approximate the desired slot contents as good as possible. As identical commodity combinations may appear in different slots and each commodity combination corresponds to a set of orders, this yields a set of preferred order slots for each order.

The problem of computing preferred order slots is related to the multidimensional version of the well-known bin packing problem (see [GJ79]). In our case, however, the number of bins is fixed, the desired bin content is known, and bins are thus allowed to be overloaded.

Problem 9.1 Best Vector Packing into Bins Problem (BVPBP)

Instance *A set B of bins, a desired bin content $w_b^* \in \mathbb{R}^d_{\geq 0}$ for each $b \in B$, and a set I of vectors $w_i \in \mathbb{R}^d_{\geq 0}$ for which $\sum_{b \in B} w_b^* = \sum_{i \in I} w_i$ holds.*

Question *Approximate each w_b^* with elements of I in an optimal way, i. e. find a packing $P : I \to B$ such that $\sum_{b \in B} ||w_b^* - \sum_{i \in P^{-1}(b)} w_i||^2$ is minimized.*

To our knowledge, the BVPBP is a new combinatorial problem which is not dealt with in the literature yet. Due to both a tight project schedule and running time reasons, we had to do without an exact solution algorithm for the BVPBP. We

Algorithm 9.1 A heuristic for the solution of the BVPBP.

Set $A(b) = \emptyset$ for all bins $b \in B$
Define $\Gamma(b) := w_b^* - \sum_{a \in A(b)} w_a$ for $b \in B$

while $I \neq \emptyset$ **do**
Choose a vector $w_{i_0} \in I$ and a bin $b_0 \in B$ such that

$$||\Gamma(b_0) + w_{i_0}||^2 - ||\Gamma(b_0)||^2 \to \min!$$

Set $A(b_0) = A(b_0) \cup w_{i_0}$
Set $I = I \setminus w_{i_0}$
return the set $A(b)$ of assigned vectors for each bin $b \in B$

applied the greedy heuristic depicted in Algorithm 9.1 instead, which chooses a vector and a bin at each step so that the vector improves on the bins contents best possible.

An instance of the BVPBP in the context of order clustering consists of the set of slots $B = \{1, \ldots, S\}$ and vectors for the desired slot contents which respect banning rules as well as vectors whose components arise from commodity combinations which respect clustering and spreading rules. We consider the partition $\mathcal{P}(\mathcal{C}_{\text{Clu}}) = [o]_1 \cup \ldots \cup [o]_u$ of the order set $\mathcal{O}$ and set $r := |\mathcal{C}_{\text{Spr}}|$ and introduce a vector for each commodity combination in $\mathcal{P}(\mathcal{C}_{\text{Clu}})$.

Definition 9.11 *Let $[o]_i \in \mathcal{P}(\mathcal{C}_{Clu})$ be a commodity combination. The vector $w_i = (w_{i1}, \ldots, w_{ir}, w_{i,r+1}) \in \mathbb{N}_{\geq 0}^{r+1}$ consists of weights defined by*

$$w_{ij} := \begin{cases} |[o]_i \cap \mathcal{O}(c_j)| & , \text{for } j = 1, \ldots, r \\ |[o]_i| & , \text{for } j = r+1 \end{cases}$$

for $i = 1, \ldots, u$ and $c_j \in \mathcal{C}_{Spr}$.

As $\mathcal{P}(\mathcal{C}_{\text{Clu}})$ is a partition, it holds $\sum_{i=1}^{u} w_{ij} = |\mathcal{O}(c_j)|$ for $j = 1, \ldots, r$. Recall that each set $[o]_i \in \mathcal{P}(\mathcal{C}_{\text{Clu}})$ is affected by at most one clustering rule (see Section 9.1.7). If a clustering rule affects a set $[o]_i$ and allows its contents to be clustered in $s > 1$ slots, the corresponding vector w_i is replaced by s new vectors with weights $\frac{w_{ij}}{s}$. For example, the vector that corresponds to the commodity combination for which no clustering rule holds is always split into S vectors, i. e. into as many vectors as there are slots of the master zone.

Definition 9.12 *The* desired slot contents $w_s^* = (w_{s1}^*, \ldots, w_{sr}^*, w_{s,r+1}^*) \in \mathbb{N}_{\geq 0}^{r+1}$ *of slot s of the master zone consists of weights defined by*

$$w_{sj}^* := \begin{cases} \frac{|\mathcal{O}(c_j)|}{S} \cdot b(c_j, z_m; s) & , \text{for } j = 1, \ldots, r \\ \frac{n}{S} & , \text{for } j = r+1 \end{cases}$$

for $s = 1, \ldots, S$ and $c_j \in \mathcal{C}_{Spr}$, where $b(c_j, z_m; s)$ denotes the banning factor for slot s as introduced in Section 9.1.7.

Note that, in case of absence of any banning rule, an enumerative approach to the BVPBP can be accelerated by the use of methods similar to those described in Section 7.2.2 and that $\sum_{i=1}^{u} w_{i,r+1} = \sum_{s=1}^{S} w^*_{s,r+1} = n$ holds, i. e. we introduce the $(r+1)$-th component of each vector to achieve evenly occupied slots. As each vector w_i corresponds to a commodity combination $[o]_i \in \mathcal{P}(\mathcal{C}_{\text{Clu}})$ which in turn corresponds to a set of orders, the result of Algorithm 9.1, if applied to the set of slots and vectors as defined in Definition 9.11 and Definition 9.12, is an ambiguous suggestion how to assign orders to slots of the master sequence.

Definition 9.13 *We denote the set of preferred slots for an order as suggested by Algorithm 9.1 by*

$$S^*(o) = \{s \in \{1, \ldots, S\} : \exists w_i \in A(s) \text{ such that } o \in [c]_i\}$$

for $o \in \mathcal{O}$.

9.3.2 Assignment of orders to slots

The preferred order slots suggested by Algorithm 9.1 yield no usable assignment of orders to slots in general (as there is no guarantee that all slots are filled with the same number of orders and there may be more than one preferred slot for an order), but can be used to compute an actual assignment of orders to slots by a linear programming approach.

Definition 9.14 *We denote the set of commodity combinations affected by a commodity $c \in \mathcal{C}$ by $P(c) := \{[o]_i \in \mathcal{P}(C) : c \in o \text{ for } o \in [o]_i\}$, where $\mathcal{P}(C)$ is a partition of the order set $\mathcal{O}$.*

We consider the partition of $\mathcal{O}$ induced by the set of banning, clustering, and spreading commodities and assume that $\mathcal{P}(\mathcal{C}_{\text{Ban}} \cup \mathcal{C}_{\text{Clu}} \cup \mathcal{C}_{\text{Spr}}) = [o]_1 \cup \ldots \cup [o]_q$. We introduce a variable x_{is} for each commodity combination $i = 1, \ldots, q$ and each slot $s = 1, \ldots, S$ that counts the number of orders in $[o]_i$ to be assigned to slot s. The value of each x_{is} has to respect the following constraints and objectives.

Slot size constraint The number of orders in each slot must clearly not exceed the maximal slot size. Therefore,

$$\sum_{i=1}^{q} x_{is} = \frac{n}{S}$$

must hold for all slots $s = 1, \ldots, S$.

Banning constraint Recall that $B(c, z_m; s)$ denotes the set of banned sequence positions for a commodity $c \in \mathcal{C}$ from the master zone $z_m \in \mathcal{Z}$ in slot s. As we must not exceed the number of allowed sequence positions for c in the master zone z_m in slot s,

$$\sum_{[o]_i \in P(c)} x_{is} \leq \frac{n}{S} - |B(c, z_m; s)|$$

must hold for all slots $s = 1, \ldots, S$ and commodities $c \in \mathcal{C}_{\text{Ban}}$.

Clustering objective The occurrence of an order $o \in \mathcal{O}$ within a slot $s \notin S^*(o)$ has to be avoided. Therefore, we demand

$$\sum_{[o]_i \in P(c)} x_{is} \geq 0$$

for all slots $s \notin S^*(o)$ and commodities $c \in \mathcal{C}_{\text{Clu}}$. Our objective is to reach equality for these constraints.

Spreading objective The spreading of commodities requires the consideration of banning rules. As the number of orders with a spreading commodity within each slot must therefore be scaled with the banning factor,

$$\sum_{[o]_i \in P(c)} x_{is} = \frac{|\mathcal{O}(c)|}{S} \cdot b(c, z_m; s)$$

must hold for all slots $s = 1, \ldots, S$ and commodities $c \in \mathcal{C}_{\text{Spr}}$. Our objective is to keep equality for these constraints.

We derive an integer program from these constraints and objectives by the introduction of slack variables y_{is}^+, y_{is}^-, and z_{is} whose sum is to be minimized in its objective function. The linear relaxation of the integer program is depicted in Figure 9.3.

Note that we assume that $\mathcal{C}_{\text{Clu}} \cap \mathcal{C}_{\text{Spr}} = \emptyset$ holds, as the restrictions arising from clustering and spreading objectives are inconsistent otherwise. We solve the linear program depicted in Figure 9.3 successively for all rules from priority $p = 1$ up to priority $p = 10$ to achieve a strict separation of rules of different priorities. The first run considers only rules of priority $p = 1$ and its solution yields an initial set of constraints that are taken over to the next run. The second run includes rules of priority $p = 2$ as well and refines the solution of the first run, as the consideration of additional commodities yields a refinement of commodity combinations. Again, additional constraints are taken over to next run, until all rules and a variety of accumulated constraints of previous runs are considered in the final run.

$$
\begin{array}{rcl}
\sum_{i=1}^{q}\sum_{s=1}^{S}(z_{is}+y_{is}^{+}+y_{is}^{-}) & \rightarrow & \text{min!} \\
\sum_{i=1}^{q} x_{is} & = & \frac{n}{S} \quad \forall s=1,\ldots,S \\
\sum_{[o]_i\in P(c)} x_{is} & \leq & \frac{n}{S}-|B(c,z_m;s)| \quad \forall s=1,\ldots,S \quad \forall c\in\mathcal{C}_{\text{Ban}} \\
\sum_{[o]_i\in P(c)}(x_{is}-z_{is}) & = & 0 \quad \forall s\notin S^{*}(o) \quad \forall c\in\mathcal{C}_{\text{Clu}} \\
\sum_{[o]_i\in P(c)}(x_{is}+y_{is}^{+}-y_{is}^{-}) & = & \frac{|\mathcal{O}(c)|}{S}\cdot b(c,z_m;s) \quad \forall s=1,\ldots,S \quad \forall c\in\mathcal{C}_{\text{Spr}} \\
x_{is} & \geq & 0 \quad \forall i=1,\ldots,q \quad \forall s=1,\ldots,S \\
z_{is} & \geq & 0 \quad \forall i=1,\ldots,q \quad \forall s=1,\ldots,S \\
y_{is}^{+} & \geq & 0 \quad \forall i=1,\ldots,q \quad \forall s=1,\ldots,S \\
y_{is}^{-} & \geq & 0 \quad \forall i=1,\ldots,q \quad \forall s=1,\ldots,S
\end{array}
$$

Figure 9.3: The linear program for the assignment of orders to slots.

The result of these iterations is a set of values x_{is} that suggests to assign x_{is} orders of $[o]_i$ to slot s. The actual assignment is in the end computed by a straightforward greedy picking algorithm that decides which orders of a commodity combination are assigned to a slot and additionally resolves the problem of rounding the values x_{is} to suitable integer values. Furthermore, it has to provide a strategy to cope with situations in which the linear program in Figure 9.3 is infeasible, although we can almost always assume a reasonable input for the linear program in practice.

9.3.3 Computation of spreading distances

After the computation of preferred order slots and the assignment of orders to slots, we can easily deduce the number of occurrences of any commodity $c \in \mathcal{C}$ within each slot $s \in \{1,\ldots,S\}$ of the master zone, which we use to prepare the evaluation of spreading rules. Recall that we may always evaluate spreading rules in the master zone instead of the zone specified by the spreading rule.

Definition 9.15 *We denote the* desired average spreading distance *for a commodity $c \in \mathcal{C}_{Spr}$ in the master zone by $A(c,z_m;s)$ for each slot $s=1,\ldots,S$.*

Note that $A(c,z_m;s)$ is in general not an integer. Therefore, both $\lfloor A(c,z_m;s)\rfloor$ and $\lceil A(c,z_m;s)\rceil$ are the actually desired spreading distances.

9.4 Master sequence construction

The master sequence is constructed slot by slot in a straightforward greedy fashion as described in Algorithm 9.2. We iterate through all sequence positions $i =$

$1, \ldots, n$ of the master sequence and temporarily assign each unsequenced order to the current position i. The effect on the quality of the master sequence arising from the current assignment placement is rated by an evaluation of the rule set with respect to the commodities of the order (see Section 9.4.1). The order whose assignment is rated best is sequenced to position i and we proceed with position $i+1$. Note that it is sufficient to evaluate rules only for one order of each commodity combination and that commodity combinations may change from slot to slot.

Algorithm 9.2 The master sequence construction algorithm.

Set $i = 0$ and $M = \emptyset$

for all slots of the master zone **do**
  Initialize the set U of unsequenced orders of the current slot
  Compute all commodity combinations $[c]_1, \ldots, [c]_j$ of orders in U
  while $[c]_1 \cup \ldots \cup [c]_j \neq \emptyset$ **do**
    Set $i = i + 1$
    Choose an order $o^* \in [c]_k$ that incurs a minimal violation of rules
    Set $M_i = o^*$
    Set $[c]_k = [c]_k \setminus o^*$
return the master sequence M

We remark that it is possible to support the efficiency of Algorithm 9.2 in several ways. For example, enamel color clustering can be supported by sorting the orders within each commodity combination with respect to their enamel color (if clustering rules affect enamel colors only). This increases the probability that identical colors are sequenced close to each other, which in turn increases the probability that larger color blocks can be built by short-term color interchanges in front of the paint shop.

9.4.1 Rule evaluation

Due to the deterministic model of the production process described by the framework in Section 9.1, the master sequence determines the order sequence in all zones of the plant. Whenever an order is assigned to a position of the master sequence, we can therefore check the quality of the assignment with respect to the given rule set. Note that we rate only a single assignment rather than a set of assignments.

Definition 9.16 *Let $o \in \mathcal{O}$ be an order sequenced to position i of the master sequence. The resulting* number of rule breaches *are collected in a vector $b = b(o, i) \in \mathbb{N}_{\geq 0}^{10}$, where $b_p = r$ if and only if the sequencing of order o to position*

i of the master sequence causes r rules of priority p to be broken. Likewise, a vector $v = v(o,i) \in \mathbb{R}^{10}$ of penalty values *is created, where $v_p = r$ if and only if the sequencing of order o to position i of the master sequence causes the resulting penalty values for all rules of priority p to sum up to the value r.*

Each rule can be broken at most once for each commodity. For each broken rule of priority p, either the number of rule breaches b_p or the penalty value v_p or both are updated.

In the following, we describe the number of rule breaches and penalty values resulting from broken rules, assuming that an order $o \in \mathcal{O}$ is sequenced to position $i \in \{1,\dots,n\}$ in slot $s \in \{1,\dots,S\}$ of the master sequence and that $c \in \mathcal{C}$ resp. $z \in \mathcal{Z}$ denotes the commodity resp. the zone affected by a rule of priority p. Recall the notation introduced in Section 9.1.7 and that $P(o,i,z)$ denotes the position of o in z (see Definition 9.7). We express rule breaks and penalty values in terms of commodity distances to keep them comparable, with the exception of spreading rules.

Banning rule Let $l := \min\{i : i \in B(c,z)\}$ resp. $u := \max\{i : i \in B(c,z)\}$ denote the minimal resp. the maximal banned position for c in z. If $c \in o$ and $l \le P(o,i,z) \le u$, the number of rule breaches b_p is increased by 1 and the penalty value is increased by $v_p = 1 + \min\{P(o,i,z) - l, u - P(o,i,z)\}$.

Clustering rule The clustering rule is not evaluated, as the order clustering computed in Section 9.3 is not allowed to be changed.

Grouping rule Let

$$g := \max\{j : P(o,i,z) - j \text{ holds an order } o' \text{ with } c \in o' \text{ for all } k = 1,\dots,j\}$$

denote the maximal length of a consecutive subsequence of orders in z that have commodity c, starting from $P(o,i,z)$-1.

1. If $g > 0$ and $c \in o$, the rule is assumed to be broken if $g+1 > g_{\max}(c,z)$.
2. If $g > 0$ and $c \notin o$, the rule is assumed to be broken if $g < g_{\min}(c,z)$.
3. If $g = 0$ and $c \in o$, let

$$d := \min\{j : P(o,i,z) - j \text{ holds an order } o' \text{ with } c \in o'\}$$

denote the minimal distance of o in z to an order that has commodity c. The rule is assumed to be broken if $d < d(c,z)$.

For any rule break, the number of rule breaches b_p remains unchanged and the penalty value is increased by $v_p = (g + 1 - g_{\max}(c,z))^2$, or $v_p = (g - g_{\min}(c,z))^2$, or $v_p = (d(c,z) - d)^2$, respectively. Note that the grouping rule is the only rule which is evaluated regardless whether $c \in o$ or $c \notin o$.

Ratio rule Let

$$x' := |\{j \in \{0, \ldots, y-1\} : P(o,i,z) - j \text{ holds an order } o' \text{ with } c \in o'\}|$$

denote the number of orders within a consecutive subsequence of orders of length y in z that have commodity c. If $x' > x$, the number of rule breaches b_p is increased by 1 and the penalty value is increased by $v_p = (x' - x)^2$.

Spacing rule Let

$$d := \min\{j : P(o,i,z) - j \text{ holds an order } o' \text{ with } c \in o'\}$$

denote the minimal distance of o to an order that has commodity c. If $c \in o$ and $d < S(c,z)$, the number of rule breaches b_p is increased by 1 and the penalty value is increased by $v_p = (S(c,z) - d)^2$.

Spreading rule A spreading rule is evaluated in the master sequence and uses the desired spreading distance $a := A(c, z_m; s)$ introduced in Section 9.3.3. Let

$$p := |\{j : P(o,i,z_m) - j \text{ is contained in slot } s\}|$$

denote the number of positions of slot s which already hold an order and let $s_e := \frac{p}{a}$ denote the expected number of orders with commodity c in slot s up to position i. Furthermore, let

$$s_f := |\{j : P(o,i,z_m) - j \text{ holds an order } o' \text{ in slot } s \text{ with } c \in o'\}|$$

denote the number of orders with commodity c actually found in slot s up to position i. If $c \in o$ and $s_e \neq s_f$, the number of rule breaches b_p remains unchanged and the penalty value is increased by $v_p = s_f - s_e$.

This way of spreading rule evaluation is also used in the well-known goal chasing algorithm (see [Mon98]), which is a standard algorithm for smooth production sequencing. Note that there is only little information for a sensible evaluation of the spreading rule at the start of each slot. This drawback can be overcome and the transition between slots can be improved by the introduction of suitable distance adjustments.

Note that it is sufficient to consider only orders sequenced to positions $p \in \{1, \ldots, P(o,i,z)\}$ resp. $p \in \{1, \ldots, P(o,i,z_m)\}$ for all rule evaluations, as the master sequence is constructed order by order within each slot.

We complete the description of Algorithm 9.2 by a description of our strategy to rate the quality of two different order assignments to the same master sequence position, using the resulting number of rule breaches and penalty values.

Definition 9.17 *Let o_1 resp. o_2 be orders sequenced to position i of the master sequence and let $b_1 = b(o_1, i)$ and $b_2 = b(o_2, i)$ resp. $v_1 = v(o_1, i)$ and $v_2 = v(o_2, i)$ denote the vector of numbers of rule breaches resp. penalty values resulting from the evaluation of all rules that affect o_1 and o_2.*

1. *If $b_1 < b_2$ with respect to lexicographical ordering, o_1 is rated better than o_2.*
2. *If $b_1 = b_2$, o_1 is rated better than o_2 if $\sum_{p=1}^{10}(11-p)(v_{1p} - v_{2p}) < 0$ holds.*
3. *If o_1 and o_2 are still rated equally, we rate o_1 better than o_2 if more rules apply to o_1 than to o_2. Otherwise, we resolve the tie between o_1 and o_2 arbitrarily.*

9.5 Computational results

Our solution approach is currently used for order sequencing in all automobile plants of the Ford Motor Company across Europe. Due to hardware variations, there has been a slight improvement in some plants, while the average color block size in the paint shop has increased from about 1.5 to about 2.2 in other plants. Considering the facts given in Section 1.2, we may therefore expect savings up to about 500,000 € per year within a plant.

The algorithms presented in Section 9.3 and Section 9.4 are imbedded into a graphical user interface provided by the Ford Motor Company, which offers an easy way to specify rules for a forthcoming production day. Additionally, it includes diagnostic tools for past production days, which in particular allow the rating of the quality of the corresponding production sequences with respect to the specified rules. Figure 9.4 – Figure 9.9 illustrate the efficiency of our approach by screenshots of this interface.

The screenshots are taken from different sequencer runs on real-world data for the plant in Cologne, Germany. The order set $\mathcal{O}$ contains $n = 1,712$ orders. The average running time for a complete sequencer run on 1,500 orders is about 3 minutes on a computer with a Sun E4500 twin processor with 450 Mhz and 1 GB memory. Figure 9.5 – Figure 9.8 illustrate that grouping, spreading, clustering, and banning rules can be expected to work efficiently for high priorities. Figure 9.9 shows the effect of Definition 9.17, which neglects commodities with low priorities.

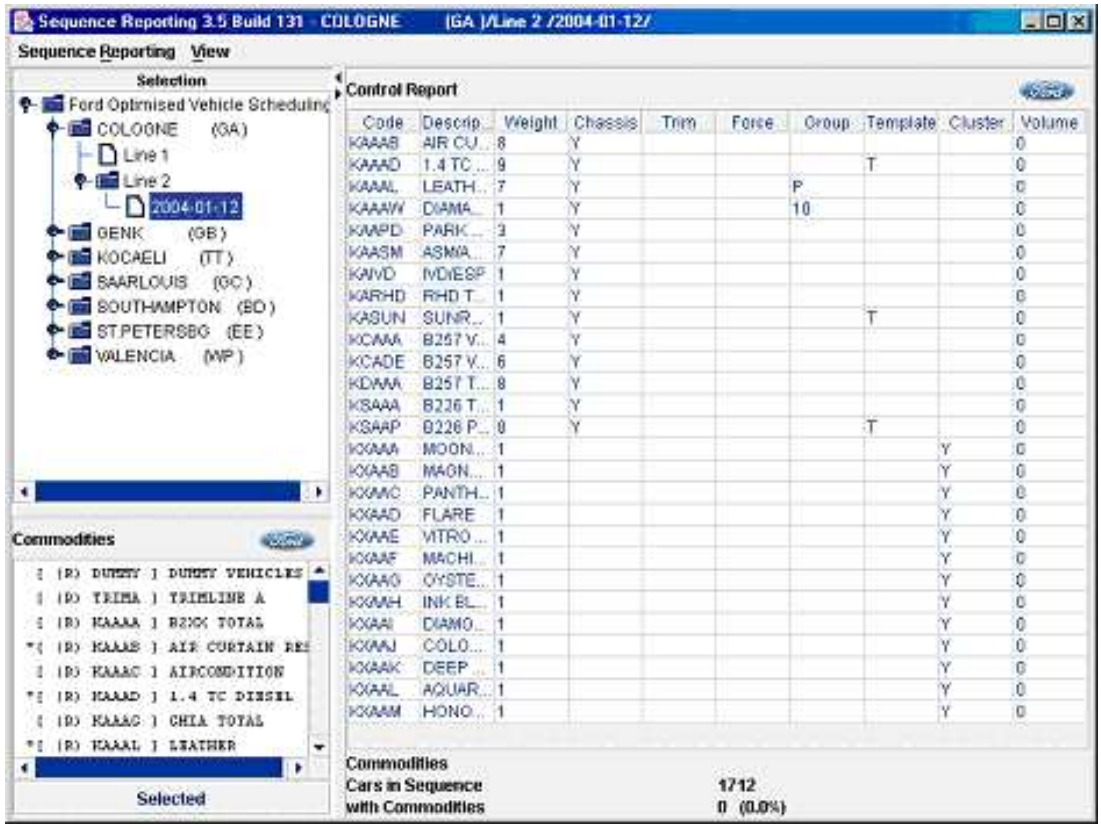


Figure 9.4: Report showing an example of rules used in a sequencer run.

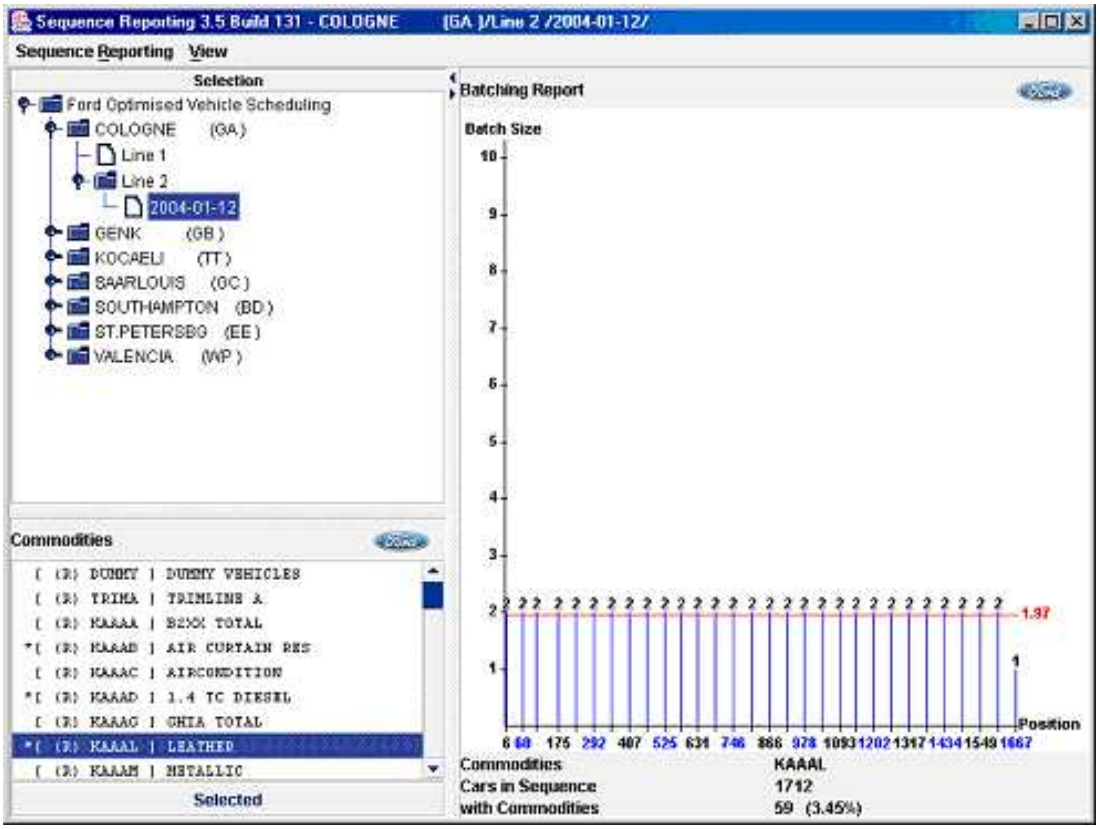


Figure 9.5: Report showing how groups of size 2 of commodity KAAAL are spread equally within the master sequence.

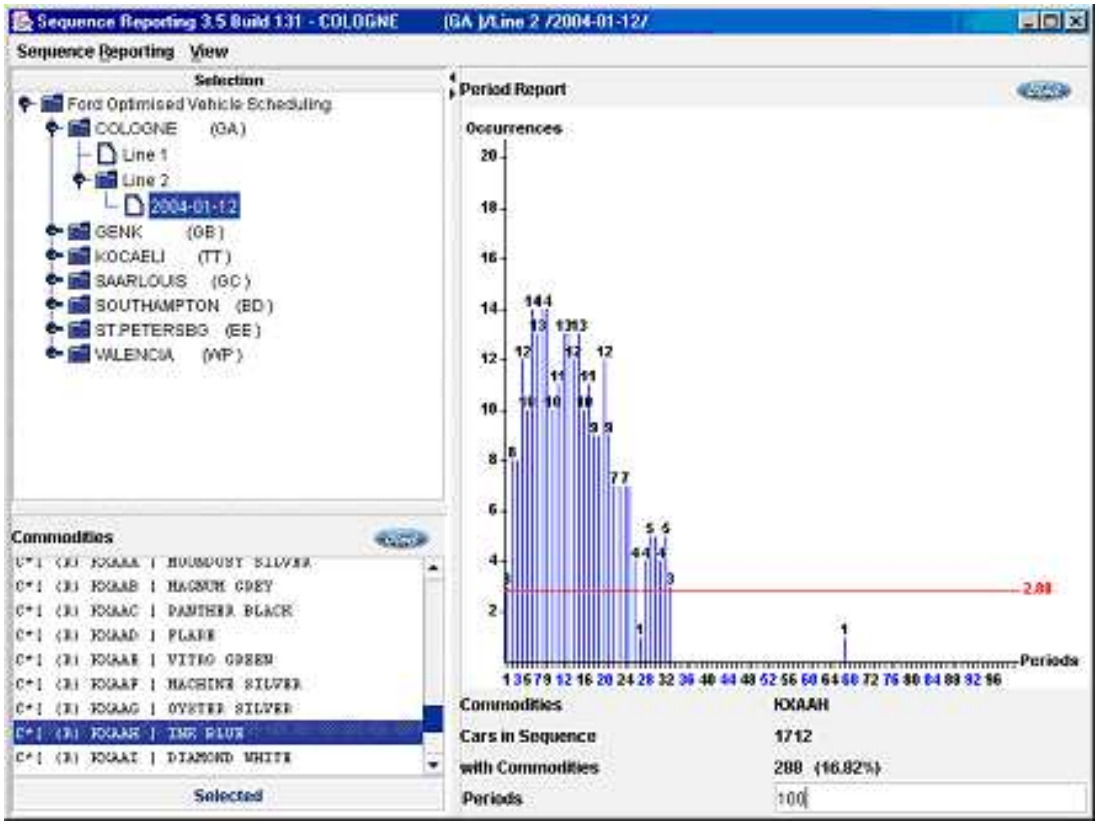


Figure 9.6: Report showing how commodity KXAAH (an enamel color) is clustered into the first slot of the master sequence.

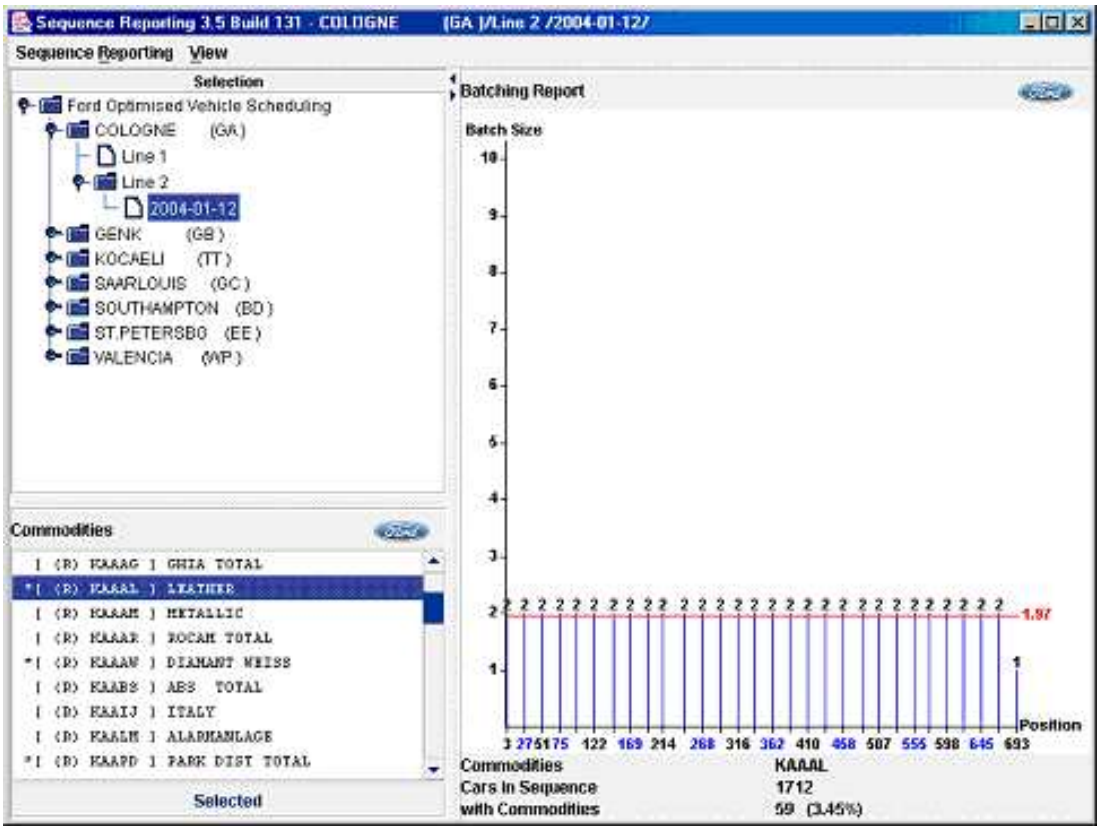


Figure 9.7: Report showing how commodity KAAAL is being banned from the last 1,000 positions of the master sequence. The last occurrence is at position 693.

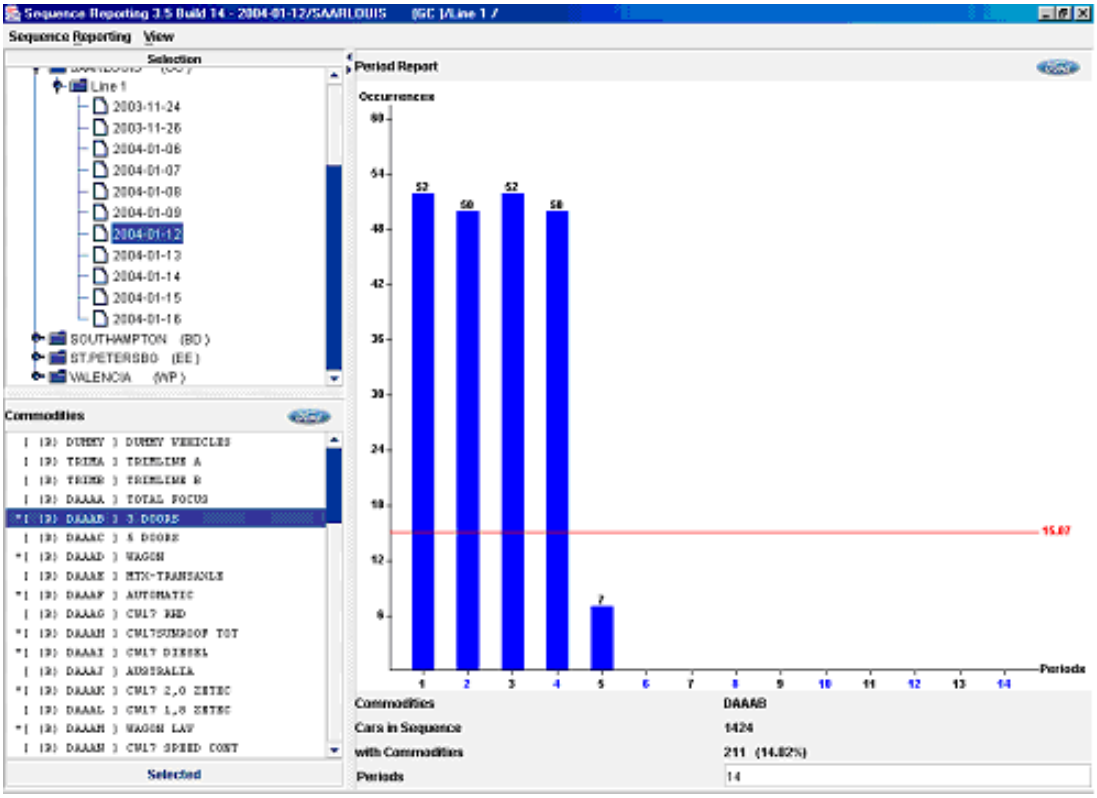


Figure 9.8: Report showing how commodity DAAAB is being banned from the last 1,000 positions of the master sequence, which is split into 14 periods of equal size. The bars show how many times DAAAB occurs within each period.

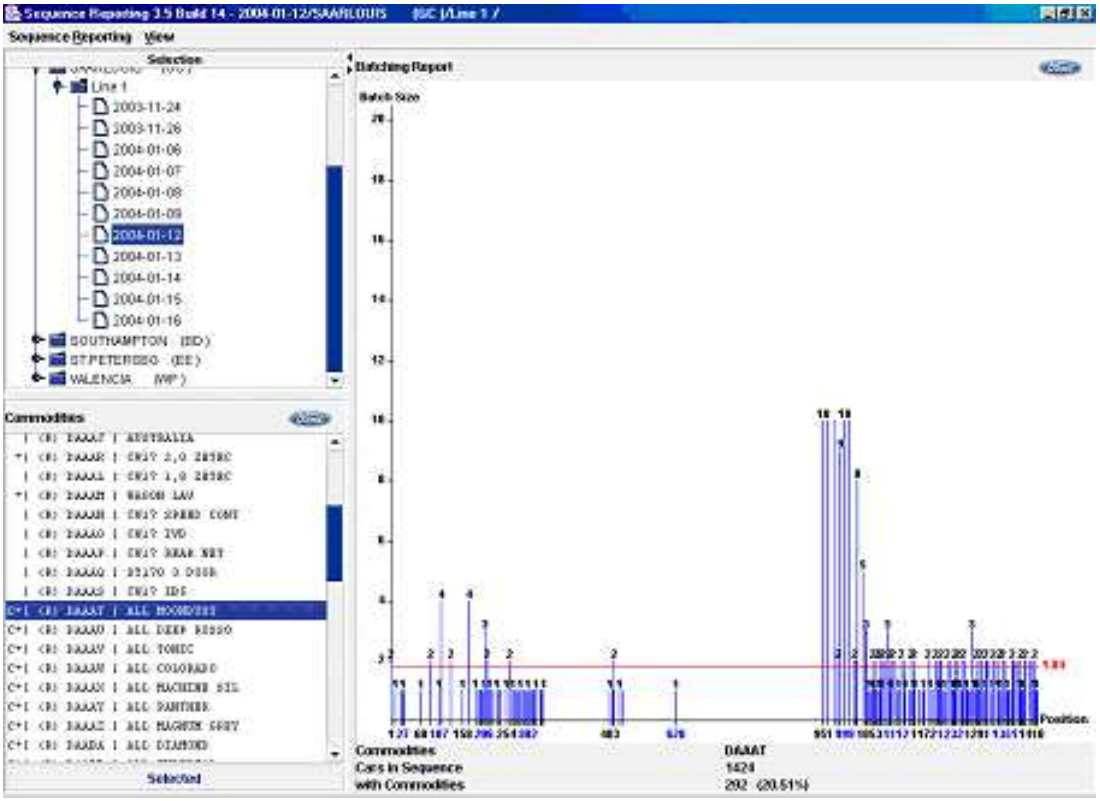


Figure 9.9: Report showing how groups of size 10 of commodity DAAAT are spread within the master sequence. Rules of higher priority prevent the grouping rule from functioning well.

Part V

Summary

The European automobile industry is faced with an increasing need for reduction of production costs to maintain its competitiveness. One contribution to significant savings is the appliance of the build-to-order policy, which does not require expensive depots. This policy, however, contrasts with an increasing customer demand for a rich product variety. Despite almost daily changing order compilations, the production process should be cost-effective.

This thesis focuses on one phase of the production process and considers three variants of a coloring problem which arises in the paint shop of an automobile plant, where each car body gets its designated enamel color. As each color change requires the spray robots to be cleaned (which in turn increases production costs), we consider the problem of minimizing the number of color changes within the enamel booth of the paint shop. Each variant of this color change minimization problem can be found in practice. The respective solution approaches divide this thesis naturally in three parts, which contain theoretical results, solution algorithms, and the description of a real-world application.

The background of the first part is the possibility of short-term interchanges of the enamel color between otherwise identical models that the automobile industry is currently looking into. The representation of each model by a letter of an alphabet yields the problem of minimizing the number of color changes within a given word, where color interchanges are allowed between identical letters only. In collaboration with Peter Oertel, Marco E. Lübbecke, and Paul S. Bonsma, we give a complexity classification of this new combinatorial new problem. In particular, we show that the problem is $\mathcal{NP}$-complete in its general formulation even if the word contains either only two colors or two letters. If both the number of colors and letters are fixed, the problem can be solved by dynamic programming. From the general case, we turn our focus to structured instances and derive lower and upper bounds on the minimal number of color changes. Indeed, we show that even a very restricted version of the problem is still $\mathcal{APX}$-hard. We reformulate this restricted version as a shortest path problem in a special class of binary matroids. As a consequence, we show that instances which do not contain both an F_7 and F_7^* minor are solvable in polynomial time. In particular, regular instances are solvable by a linear linear program. For instances which are duals of MaxFlow-MinCut matroids, we discuss consequences of the dualized MaxFlow-MinCut theory.

The second part provides algorithmic approaches for a color change minimization strategy that uses a line storage system installed in front of the paint shop. Line storage systems are a common tool in the automobile industry, as they provide an easy and cost-effective way to build color blocks for the paint shop by a suitable color storage and retrieval strategy. We consider two offline strategies and one online strategy that may be applied to a line storage system for a color change minimization in the paint shop. The first offline strategy considers the retrieval

of colors only and adapts a dynamic program for the multiple sequence alignment problem from molecular biology. The result is a new and exact solution algorithm which, although the multiple sequence alignment problem is $\mathcal{NP}$-complete, turns out to be highly efficient for typical real-world instances. The second offline strategy is a generalization of the first and considers both the storage and retrieval of colors. The colors that have to be stored on the line storage system are held in a lookahead area. We present a dynamic program that enumerates all feasible states of the line storage system by the use of two basic operations. The resulting algorithm has, however, a huge theoretical complexity and is, even after employing several methods which drastically accelerate the running time of an implementation, only applicable to small instances. Nevertheless, it is not only of theoretical interest as we use parts of it for an online color storage and retrieval simulation. We employ only one basic operation this time, which is modelled more closely on practice. While one half of the lookahead area is processed, an optimal color storage and retrieval strategy is computed for the second. These partial solutions can be assembled into a solution for a complete production day, and even a greedy assembly leads to a significant increase of color block sizes in the paint shop. In particular, the simulation can be adapted easily to practical requirements like memory usage and the observance of production cycle times. We give computational results on randomly created instances for all algorithms which accompany discussions of their pros and cons.

The description of a real-world project in cooperation with the Ford Motor Company is the content of the third part. The color change minimization strategy is similar to the strategy of short-term color interchanges of the first part. We consider, however, the additional fact that most automobile plants have not only one, but usually two or three enamel booths arranged parallel to each other. This enables us to build color blocks within each enamel booth by a suitable routing of orders, if identical enamel colors arrive close to each other at the split point to the enamel booths. This is of particular importance as unavoidable manufacturing errors prevent the maintenance of color blocks during the production process. Our solution approach extends known solution approaches in several ways. In particular, its scope is not restricted to the paint shop but includes the complete production process, which leads to several additional optimization objectives and constraints for other production phases that have to be respected. The possibility of a suitable order routing is taken into account by a clustering step that leads to an accumulation of orders with identical enamel colors within specific parts of the production sequence. The unavoidable occurrence of manufacturing errors is anticipated by the introduction of deterministic order delays. The combination of these and other concepts yields an efficient solution algorithm, which is currently used in all plants of the Ford Motor Company across Europe.

Bibliography

[A+98] Sanjeev Arora et al. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[AK00] Paola Alimonti and Viggo Kann. Some $\mathcal{APX}$-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1–2):123–134, 2000.

[AKG] `http://www.akg.de/technik/text/239051.html` (visited in March 2003).

[BEH03] Paul S. Bonsma, Thomas Epping, and Winfried Hochstättler. Complexity results on restricted instances of a paint shop problem for words. *Discrete Applied Mathematics*, 2003. Submitted.

[Ber01] Michael E. Bergen. *Constraint-based assembly line sequencing.* PhD thesis, University of Alberta, 2001.

[Cam65] Paul Camion. Characterization of totally unimodular matrices. *Journal of the AMS*, 16:1068–1073, 1965.

[EH03a] Thomas Epping and Winfried Hochstättler. Shortest paths through two-tone pairs. *Electronic Notes in Discrete Mathematics*, 13, 2003. Extended abstract.

[EH03b] Thomas Epping and Winfried Hochstättler. Sorting with line storage systems. In Ulrike Leopold-Wildburger, Franz Rendl, and Gerhard Wäscher, editors, *Operations Research Proceedings 2002*, pages 235–240, Berlin, 2003. Springer. Extended abstract.

[EHL03] Thomas Epping, Winfried Hochstättler, and Marco E. Lübbecke. Maxflow-mincut duality for a paint shop problem. In Ulrike Leopold-Wildburger, Franz Rendl, and Gerhard Wäscher, editors, *Operations Research Proceedings 2002*, pages 377–382, Berlin, 2003. Springer. Extended abstract.

[EHO01] Thomas Epping, Winfried Hochstättler, and Peter Oertel. Some results on a paint shop problem for words. *Electronic Notes in Discrete Mathematics*, 8, 2001. Extended abstract.

[EHO04] Thomas Epping, Winfried Hochstättler, and Peter Oertel. Complexity results on a paint shop problem for words. *Discrete Applied Mathematics*, 136(2–3):217–226, 2004. Special issue: The 1st Cologne-Twente workshop on graphs and combinatorial optimization.

[GGL95] Ronald L. Graham, Martin Grötschel, and Laszlo Lovasz, editors. *Handbook of Combinatorics (Volume II)*. Elsevier, 1995.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and intractability: A guide to the theory of $\mathcal{NP}$-completeness*. Freeman, 1979.

[Gol80] Martin C. Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, 1980.

[Hac00] Rainer Hackl. *Optimierung von Reihenfolgeproblemen mit Hilfe Genetischer Algorithmen*. PhD thesis, Universität Regensburg, 2000.

[Hli] Petr Hlineny. *The MACEK (Matroids Also Computed Efficiently Kit) project*. `http://www.mcs.vuw.ac.nz/research/macek/` (visited in June 2003).

[HTC92] Ju Yuan Hsiao, Chuan Yi Tang, and Ruay Shiung Chang. An efficient algorithm for finding a maximum weight 2-independent set on interval graphs. *Information Processing Letters*, 43:229–235, 1992.

[II94] Takahiro Ideka and Hiroshi Imai. Fast A^* algorithms for multiple sequence alignment. In *Proceedings of the Genome Informatics Workshop V*, pages 90–99. Universal Academy Press, 1994.

[Lot97] M. Lothaire. *Combinatorics on words*. Cambridge University Press, 1997.

[LR00] Martin Lermen and Knut Reinert. The practical use of the A^* algorithm for exact multiple sequence alignment. *Journal of Computational Biology*, pages 655–671, 2000.

[MN95] Kurt Mehlhorn and Stefan Näher. Leda, a platform for combinatorial and geometric computing. *Communications of the ACM*, 38(1):96–102, 1995.

[Mon98] Yasuhiro Monden. *Toyota production system: An integrated approach to just-in-time*. Engineering & Management Press, 1998.

[Oxl92] James G. Oxley. *Matroid theory*. Oxford University Press, 1992.

[PFO] `http://www.pfonline.com/articles/01sum3b.html` (visited in December 2003).

[PG02] Markus Puchta and Jens Gottlieb. Solving car sequencing problems by local optimization. In Stefano Cagnoni et al., editors, *EvoWorkshops 2002*, pages 132–142, Berlin Heidelberg, 2002. Springer.

[PY91] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

[REI] Knut Reinert. *Private communication.*

[Sey77] Paul D. Seymour. The matroids with the max-flow min-cut property. *Journal of Combinatorial Theory*, B(23):189–222, 1977.

[Spi02] Sven Spieckermann. *Neue Lösungsansätze für ausgewählte Planungsprobleme in Automobilrohbau und -lackiererei*. Shaker, 2002.

[Tru92] Klaus Truemper. *Matroid decomposition*. Academic Press, 1992. `http://www.emis.de/monographs/md/` (visited in September 2003).

[Tut58] William T. Tutte. A homotopy theorem for matroids, I, II. *Transactions of the AMS*, 88:144–174, 1958.

[VDA03] Verband der Automobilindustrie e. V. *Auto 2003 - Jahresbericht des VDA*, 2003.

[Wat95] Michael S. Waterman. *Introduction to computational biology: maps, sequences and genomes*. Chapman & Hall, 1995.

[Wil01] Horst Wildemann. *Das Just-in-Time Konzept: Produktion und Zulieferung auf Abruf*. TCW, 2001.

[WJ94] Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.

[WS95] Dirk Wortmann and Sven Spieckermann. Manufacturing line simulation of automotive industry to enhance productivity and profitability. In M. R. Heller, editor, *Automotive Simulation '95*, pages 91–106. Society for Computer Simulation International, 1995.

[ZES97] Jürgen Zimmermann, Christoph Engel, and Alfons Steinhoff. Order-sequencing in automobile production. Technical Report WIOR – 502, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe, July 1997.